UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Cálculo de altas prestaciones para la lectura automática de códigos de antibiótico en antibiogramas

Autor/es

MANUEL GARCÍA DOMÍNGUEZ

Director/es

CÉSAR DOMÍNGUEZ PÉREZ y JÓNATAN HERAS VICENTE ,

Facultad

Escuela de Máster y Doctorado de la Universidad de La Rioja

Titulación

Máster Universitario en Tecnologías Informáticas

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

Trabajo de Fin de Máster

Cálculo de altas prestaciones para la lectura automática de códigos de antibiótico en antibiogramas

Autor:

Manuel García Domínguez

Tutor/es: César Domínguez y Jónathan Heras

MÁSTER:
Máster en Tecnologías Informáticas (853M)

Escuela de Máster y Doctorado

AÑO ACADÉMICO: 2016/2017

ÍNDICE

RESUMEN

La necesidad de este proyecto viene precedida de un problema biológico tratado en otro proyecto del Grupo de Informática de la Universidad de La Rioja en conjunción con el Grupo de Ecología Molecular de la Resistencia de los Antibióticos y Seguridad Alimentaria del Departamento de Agricultura y Alimentación.

Este problema consta de hacer un análisis de un antibiograma, técnica biológica que mide la resistencia de las bacterias ante antibióticos. Hasta el momento, el grupo de ecología realizaba el estudio manualmente, con todos los problemas que esto conlleva. Para solucionar el problema se ha desarrollado un programa llamado AntibiogramJ, cuyo fin es leer semiautomáticamente los antibiogramas. Este proceso de análisis de antibiogramas consta de varios pasos, los cuales tienen más de una alternativa, tal y como sucede en el paso de la lectura de los códigos. Este proyecto se va a centrar en ese paso, en encontrar la mejor técnica posible para la lectura de los códigos que tienen grabados los discos de antibióticos. Para encontrar la mejor técnica de reconocer los códigos citados se va tener que probar un gran número de algoritmos, por lo que también serán necesarias técnicas de paralelización para optimizar el proceso.

ABSTRACT

The need of this project is preceded by a biological problem addressed in another project of the Computer Science Group of the University of La Rioja in conjunction with the Group of Molecular Ecology of Antibiotic Resistance and Food Safety of the Department of Agriculture and Food.

This problem consists of performing an antibiogram analysis, which is a biological technique to measure the resistance of bacteria to antibiotics. Until then, the ecology group carry out the study manually, with all the problems it involves. To solve the problem it has been developed a program called AntibiogramJ that is used to read semi automatically the antibiograms. This process of analyzing antibiograms consists of several steps which have more than one alternative as happens in the step of reading the codes. This project is focused on that step, on finding the best technique for reading the codes that have the discs of antibiotic. In order to find the best technique to recognize the codes, a large number of codes will have to be tested, so parallelization techniques will be necessary to optimize the process.

INTRODUCCIÓN

La necesidad de este proyecto viene precedida de un problema biológico tratado en otro proyecto del Grupo de Informática de la Universidad de La Rioja en conjunción con el Grupo de Ecología Molecular de la Resistencia de los Antibióticos y Seguridad Alimentaria del Departamento de Agricultura y Alimentación.

El trabajo de estos biólogos trata de analizar manualmente antibiogramas, que es una técnica que nos permite medir la resistencia que presenta una bacteria frente a antibióticos. Esta medición la hacen a mano y controlan si supera o no ciertos valores para indicar si el antibiótico puede vencer a la bacteria o no. Además de eso, cada antibiótico tiene una nomenclatura diferente, que es la que podemos ver dentro de cada uno de los discos para poder diferenciarlas. El problema de esto se encuentra en que todo el proceso se realiza de forma manual, por lo que implica un gran coste a nivel de tiempo y una variación en las medidas dependiendo de quién sea la persona que las lleve a cabo.

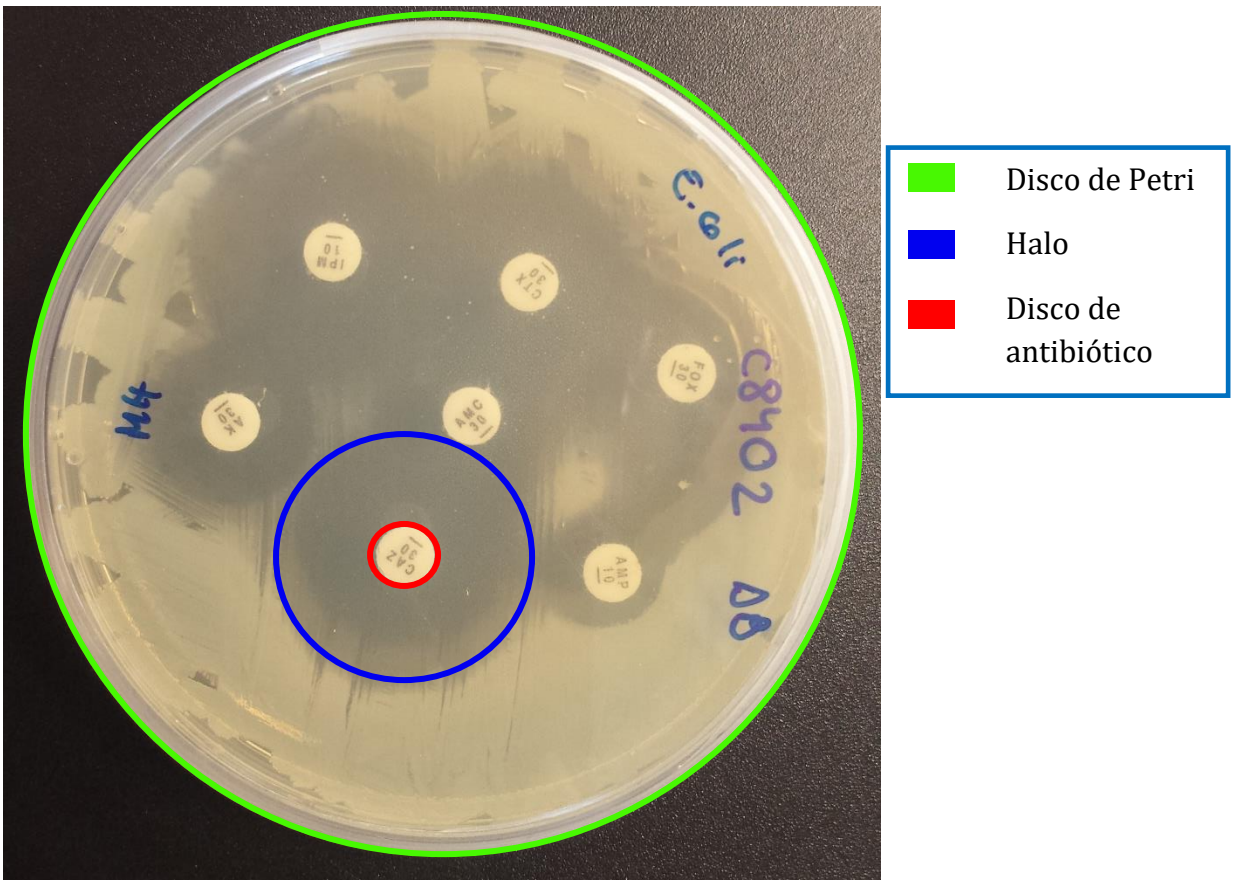

Ilustración 1. Disco de Petri

Para abordar este problema desde el Grupo de Informática de la Universidad de La Rioja se desarrolló una aplicación de escritorio, llamada AntibiogramJ (Alonso, 2017: 1-11), que permite estudiar esta situación mediante fotografías. La aplicación busca que esas imágenes puedan provenir de cualquier dispositivo con una cámara. El programa trata automáticamente la imagen, pero luego deja modificar ciertos parámetros para que los biólogos sigan teniendo cierto control sobre el resultado obtenido. La aplicación recoge los discos de antibiogramas y calcula el diámetro de los halos que podemos apreciar en la imagen (ver ilustración 1). Estos halos indican a los biólogos si el medicamento contenido en cada disco elimina o no la bacteria. Los rangos que indican esto, también son calculados y mostrados por el programa. Además de eso, reconoce el texto de cada disco y lo clasifica dependiendo del tipo de cada uno.

A lo largo de este proyecto, vamos a centrarnos en la parte dedicada al reconocimiento de los tipos de medicamentos contenidos en los discos. Se trata de reconocer el texto contenido en ellos para poder trabajar con esa información después. Para el reconocimiento de ese texto, existen muchos algoritmos diferentes por lo que es conveniente estudiarlos para determinar cuál es el que mejores resultados nos da en nuestra aplicación.

ANÁLISIS

El análisis de los antibiogramas es un proceso complejo que podemos dividir en varios pasos. Vamos a hacer una visión superficial sobre este proceso para entenderlo y nos centraremos en el paso en el cual se encuadra el objetivo de nuestro proyecto.

AntibiogramJ nos permite analizar las imágenes de los discos de Petri con ayuda de un asistente. La aplicación soporta los siguientes formatos de imagen: tiff, jpeg, png, gif, y bmp. El asistente hará un proceso guiado sobre los 5 pasos necesarios para el análisis de las imágenes de los discos de medicamentos. Los pasos en los que se divide el análisis de los discos son los que podemos ver en la imagen (ver ilustración 2) y que vamos a comentar.

El primero de ellos es el de *Bacterial isolate data* (datos de las bacterias aisladas). En este paso, los usuarios introducen la información sobre el antibiograma que se está analizando. Los datos que tiene que añadir son (entre otros)el género y la especie. De forma opcional, pueden añadir la muestra de la bacteria aislada con algunos comentarios.

El segundo paso es *Image pre-processing* (preprocesamiento de imagen). Desde la aplicación no se quiere restringir los dispositivos o las condiciones con las que se tomen la imagen del disco de Petri, haciendo que haya una gran variabilidad en las imágenes. Tanta variabilidad puede hacer que sea necesario algún ajuste en la calidad de las imágenes. El usuario puede ajustar el brillo y contraste y cortar la imagen para quedarse solo con el disco de Petri sobre el que vamos a trabajar.

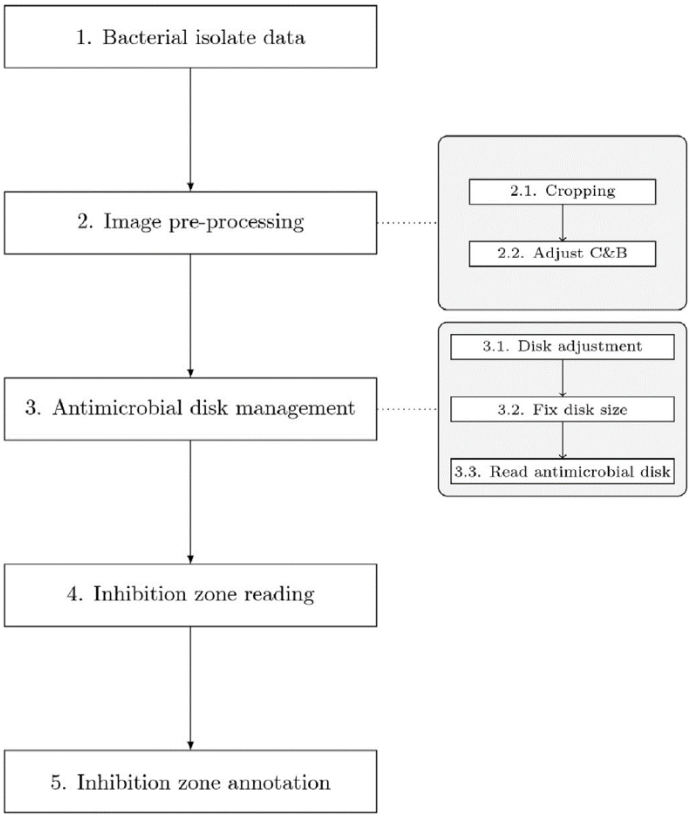

Ilustración 2. Esquema análisis de antibiogramas.

El tercer paso es *Antimicrobial disk management* (gestión de los discos antimicrobianos). La aplicación detecta automáticamente los discos de antibiótico en cada disco de Petri y lee los códigos escritos en esos discos. Para ello, se aplican una serie de filtros y umbrales que detectan los discos blancos en la imagen. En algunas ocasiones la detección automática necesita algunos ajustes. El programa le da al usuario la capacidad de añadir o borrar manualmente los discos y también añade la posibilidad de ajustar el tamaño y la posición de cada uno de ellos. Además, el usuario puede indicar el tamaño en milímetros de los diámetros de los discos detectados, que por defecto es de 6 mm. Este paso nos permite calibrar la escala de la imagen y así poder usar diferentes dispositivos, lentes o distancias. Ahora es cuando se leen los discos de antibióticos y cada código corresponde con el antimicrobiano asociado. La lectura se realiza mediante un sistema de aprendizaje que relaciona el disco contra los discos que ha visto antes con el mismo código. La primera vez que se utiliza AntibiogramJ es necesario enseñar al sistema seleccionando elementos en una lista de antimicrobianos.

El cuarto paso es *Inhibition zone reading* (lectura de la zona de inhibición). La aplicación determina y mide automáticamente las zonas de inhibición de los discos de Petri y los categoriza basándose en el diámetro y los valores límite para clasificarlos. Las categorías disponibles son: Susceptible, Intermedio, Resistente y No Disponible. El usuario puede introducir manualmente las zonas y también las puede modificar haciendo que la categoría se recalcule automáticamente. En este paso se pueden ver los límites y valores de la susceptibilidad. Con este programa podemos categorizar fácilmente cada zona de inhibición y verla en un momento haciendo que el usuario sea capaz de proporcionar una lectura interpretada del antibiograma como complemento del resultado de la prueba.

Por último, *Inhibition zone annotation* (anotación de la zona de inhibición). En este paso, el usuario puede anotar la imagen añadiendo información como la categoría y el diámetro de cada zona de inhibición y personalizarla con diferentes colores.

Una vez visto el proceso completo, el proyecto se va a centrar en el tercer punto del análisis de los discos: la lectura de los códigos de los discos. El objetivo de nuestro proyecto se va a centrar en el reconocimiento de los tipos de antibióticos. Se quiere realizar un clasificador de imágenes que nos devuelva el tipo de los discos. El objetivo principal del proyecto es realizar un estudio de los resultados que obtenemos con los

diferentes algoritmos disponibles para realizar esta tarea. El estudio de los algoritmos se centrará en el tiempo y en el grado de éxito que tenga al hacer las predicciones de las imágenes que le mandemos clasificar.

Ilustración 3. Captura de AntibiogramJ

Un objetivo indirecto que conseguimos al hacer esto es la creación del clasificador de imágenes. Podremos darle al algoritmo la imagen del disco y el programa nos devolverá el tipo de ese disco. En este clasificador se irán probando todos los algoritmos disponibles para sacar la información que necesitemos para realizar el estudio. Cuando ya hayamos hecho el estudio y se haya obtenido el mejor algoritmo para nuestro proyecto, podremos usar ese método para clasificar las imágenes de la forma más efectiva posible.

Para que quede claro cuál debe ser el funcionamiento del programa pondremos un ejemplo. Nosotros le deberemos pasar la imagen que se muestra a continuación al programa.

Una vez que le pasamos una imagen como la del ejemplo (ver ilustración 4), el programa deberá devolvernos el tipo de antibiótico de que se trata, en este caso el tipo “TOB-10”.

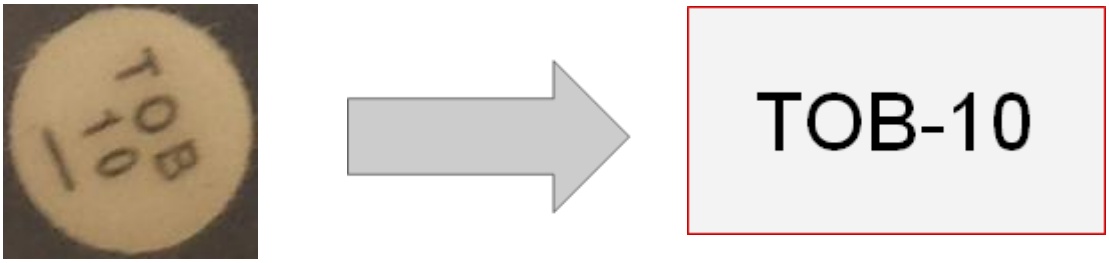

Ilustración 4. Ejemplo de funcionamiento del programa

Para poder realizar el proyecto adecuadamente, necesitamos una serie de imágenes de discos de Petri para recoger las imágenes que necesitamos.

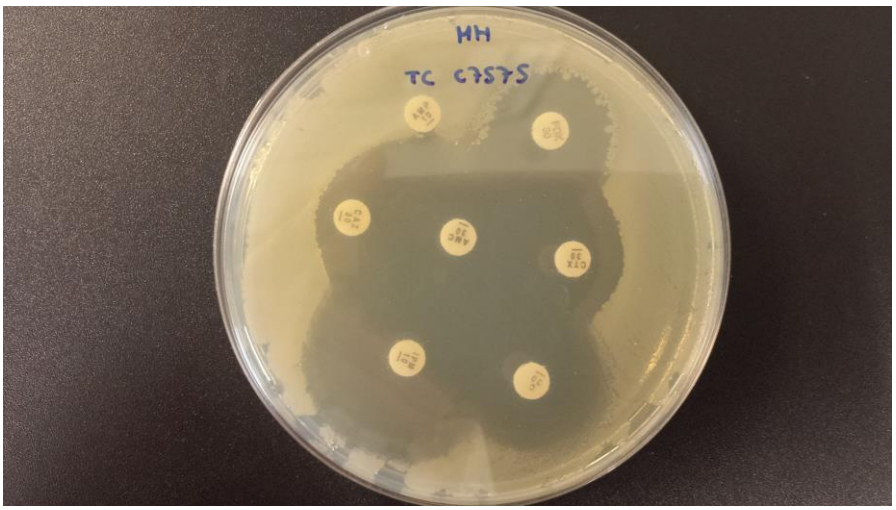

Ilustración 5. Disco de Petri

Como podemos apreciar, en la imagen de la ilustración 5 tenemos varios discos de antibióticos. Lo que tendremos que hacer es recoger cada uno de estos discos en imágenes aparte y pasarlas de una en una al programa que tenemos que desarrollar. La salida será el tipo de antibiótico que aparece en el disco que le hemos pasado. Mientras que esto para el ser humano puede resultar una tarea sencilla, veremos que lograr que un ordenador realice este mismo proceso es una tarea realmente compleja.

El estudio se deberá realizar para comprobar cuál es el algoritmo más apropiado en nuestro proyecto, ya que no existe un algoritmo que sea el mejor en todos los casos. Puede ser que un algoritmo que dé grandes resultados en otro proyecto, no sea el mejor en este. Por ello, se debe estudiar cual de todas las posibilidades es la que mejor se adapta a

nuestro problema. En la búsqueda de la mejor combinación de algoritmos, deberemos utilizar técnicas de paralelización ya que, el número combinaciones a probar será elevado y el tiempo de ejecución del programa de no usar esas técnicas puede ser inviable. Una vez que lo sepamos, podremos fijar estos parámetros y utilizar esa configuración para predecir la clase a la que pertenecen los discos que vayamos pasando por el programa.

DISEÑO

En esta sección comentaremos algunas de las decisiones tomadas para la realización del proyecto, como puede ser la de la elección de un proceso de aprendizaje frente a utilizar un reconocedor de caracteres, OCR (Optical Character Recognition), o la necesidad de utilizar el clúster de computación de Beronia.

Antes de empezar a hablar detalladamente de los pasos, tenemos que tener una idea intuitiva del proceso que deberemos seguir. Primero, se debe crear el dataset para poder trabajar con un proceso de *matching,* que consistirá en dada una imagen, encontrar la imagen de una base de datos que más se le "parece". Más tarde, tendremos que desarrollar un programa que nos dé el tipo de antibiótico que tiene escrito el disco que se le pasará como una imagen al programa. Luego, tendremos que repetir este proceso, pero con todas las imágenes del dataset para comprobar cómo funciona el algoritmo. Cuando veamos que todo funciona como se espera es cuando tenemos que comprobar el funcionamiento de todas las combinaciones de algoritmos disponibles mediante técnicas de paralelización para optimizar el tiempo de ejecución. Finalmente, después de haber realizado esto podremos obtener una conclusión de cuál es la mejor combinación para nuestro problema.

Ahora, ya podemos profundizar en algunos aspectos del proceso a realizar. El primer aspecto a comentar es que tenemos un programa parametrizado por 3 algoritmos para realizar esta tarea, este tipo de programación se conoce como programación de orden superior. Para saber las combinaciones que tenemos disponibles, tendremos que buscar qué algoritmos se podrán usar en cada uno de los parámetros que tenemos disponibles. Los parámetros que tenemos que ir combinando son:

- **Detector de puntos clave.** Con este algoritmo encontraremos las partes más significativas de una imagen. Esto depende en gran medida del algoritmo usado para detectarlos, ya que hay algoritmos que detectan como puntos clave los bordes de los objetos, otros los cambios de color o los cambios de contraste, etc.

- **Generador de descriptores.** A partir de los puntos clave recogidos por el anterior algoritmo podremos generar los descriptores que nos servirán para cuantificar y representar los objetos. Dependiendo del algoritmo usado nuevamente, los descriptores almacenarán una información diferente. Estos descriptores deben ser invariantes ante modificaciones sobre la imagen como puede ser el girarla.

- **Algoritmos de "matching".** Este algoritmo nos permitirá elegir la forma en la que queremos que los descriptores de dos imágenes se comparen para decidir, en nuestro caso, el tipo del disco de antibiótico y saber si son del mismo tipo o no.

Un ejemplo gráfico de lo que sucede lo podemos ver en la siguiente imagen (ver ilustración 6). Buscamos en ambas imágenes los puntos clave y, a partir de esos puntos clave, calculamos los descriptores mediante el algoritmo que hayamos escogido. Finalmente, relacionamos los descriptores de ambas imágenes, como se aprecia con las líneas de la imagen. Se aprecia que en el caso que las imágenes que son de la misma clase, los puntos clave relacionados entre ambas imágenes es elevado. En el segundo caso, sin embargo, vemos cómo al ser de distinta clase, los puntos clave que tienen en común las dos fotografías es bastante menor.

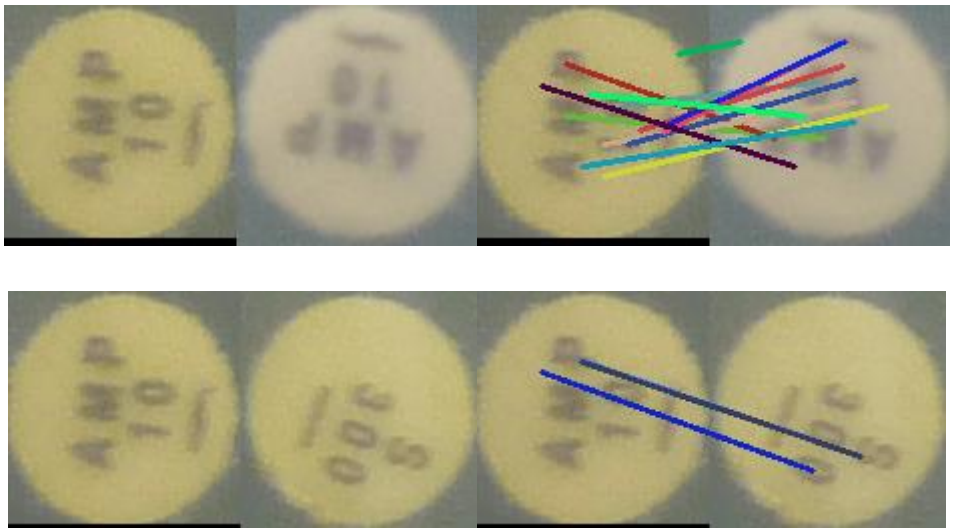

Ilustración 6. Comparación de puntos clave

Una vez visto esto, se plantean ciertas dudas como por ejemplo si es realmente necesario este proceso cuando ya sabemos que es costoso y puede ser que haya alternativas como OCR que sean más rápidas de implementar y dan buenos resultados. El realizar esta clase

de procesos conlleva mucho tiempo, hay que realizar un dataset con las imágenes disponibles y obtener nuevas imágenes a partir de las disponibles en el caso de que el dataset sea demasiado pequeño como para trabajar con él. Desarrollar el programa que nos diga el tipo predicho de la imagen que le hemos pasado. Con ello, ahora tendremos que averiguar cuál es la combinación que mejor resultado nos da en este caso.

La opción de usar el reconocimiento de caracteres se contempló, pero tenía ciertos inconvenientes. Los discos tienen que estar siempre en la misma dirección para poder leerlos y, como podemos comprobar en la imagen, cada disco tiene el texto en una dirección diferente. Otro inconveniente que se ha encontrado es que los laboratorios utilizan distintas nomenclaturas para un mismo antibiótico. Estos inconvenientes hacen que la implementación se haga más costosa y que finalmente se opte por utilizar la opción que acabamos de comentar de usar un proceso de *matching*.

Una vez que hemos elegido el proceso de *matching* tenemos que especificar qué proceso se va a llevar a cabo para que podamos encontrar la mejor combinación de los parámetros que hemos comentado. El proceso se dividirá en: creación del dataset, probar todas las posibles combinaciones de los 3 tipos de algoritmos, paralelismo y un último apartado para hacer un análisis de los resultados. Cada uno de estos pasos se verá detenidamente en la implementación del proyecto e iremos viendo conceptos que hasta ahora no han aparecido y que merecen ser comentados.

La creación del dataset es un paso muy importante con el que tendremos que entrenar el algoritmo para que el grado de éxito sea mayor. El porcentaje de acierto de los métodos depende en parte de cómo hayamos creado nuestro dataset. Si las imágenes que usamos para entrenar no son las correctas, eso se verá reflejado en el porcentaje de acierto será menor.

Para poder realizar el estudio tendremos que probar todas las posibles combinaciones de los algoritmos que se han comentado anteriormente (detectores de puntos clave, calcular los descriptores y algoritmos de matching) (ver tabla 1). Pero este proceso de probar todas las combinaciones se hace costoso debido a que probar una sola combinación ya cuesta bastante tiempo, dependiendo del tamaño del dataset (cuanto más grande sea el dataset más costará), y si además tenemos que probar todas las combinaciones posibles pues el proceso se hace inevitablemente muy largo. Entonces, hay que buscar

mecanismos para hacer que en la medida de lo posible el tiempo de ejecución se acorte al máximo, para ello la solución más fácil y a la vez más obvia que se pensó es la de hacer que el proceso se haga en paralelo.

Detector de puntos clave	Extractor de descriptores	Algoritmos de *"matching"*
StarDetector	ORB	BruteForce-Hamming
ORB	SIFT	BruteForce
AKAZE	BRISK	BruteForce-L1
Fast	AKAZE	BruteForce-Hamming(2)
MSER	Brief	FlannBased
SIFT	FREAK	
SURF	SURF	
Agast		

Tabla 1. Algoritmos disponibles

Es aquí donde debemos hablar de Beronia. Se trata de dos clústeres de servidores utilizados por la Universidad de La Rioja. Estos clústeres están compuestos por dos racks de 9 y 12 nodos respectivamente. El primer clúster consta de nodos con procesadores dodecacore; el segundo, consta de procesadores tetradecacore. Cada nodo, independientemente del rack al que pertenezca, tiene una memoria RAM de 128 GB. Además, Beronia también dispone de un disco duro de 9.7 TB en el que podemos almacenar los datos sobre los que trabajar y los resultados. Finalmente, hay un nodo de control para poder gestionar correctamente el resto de los nodos de la red. La infraestructura de los servidores es la siguiente (ver ilustración 7):

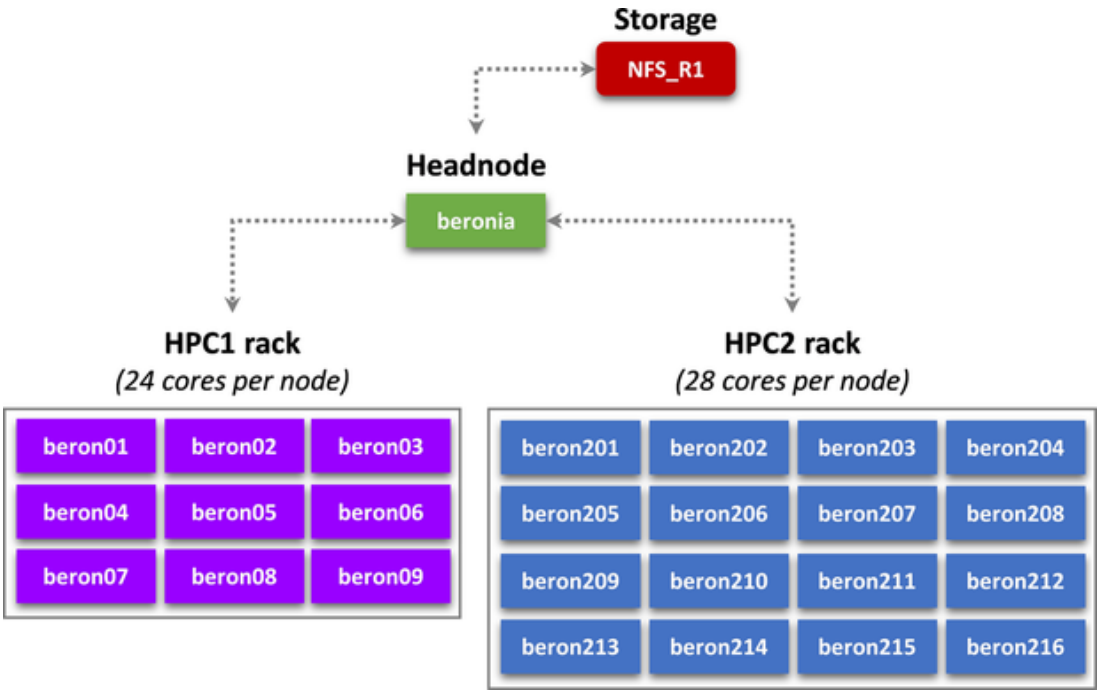

Ilustración 7. Estructura Beronia

Además de conocer las especificaciones técnicas, deberemos saber cómo mandar nuestro programa a ejecutar a este clúster. Para ello, tenemos disponible una página web con la documentación necesaria sobre Beronia, la podemos ver en el siguiente enlace: https://beroniaweb.unirioja.es/.

Gracias a que nos dan la capacidad de disponer de esta herramienta, podremos realizar la ejecución del proyecto de forma paralela ejecutándola en los servidores que acabamos de explicar. Además, al disponer de más núcleos, la ejecución en paralelo será más rápida y eficiente. Para ello, tendremos que seguir el manual que tenemos en la web de Beronia y realizar los pasos que se indican para ejecutar el programa dependiendo de las necesidades del mismo. Hay ciertas restricciones de tiempo, para que no se puedan ejecutar programas durante un mes, por ejemplo.

Una vez que hemos decidido la estructura del proyecto, podemos ponernos manos a la obra. El proyecto se dividió en varias tareas más pequeñas para poder diferenciar las partes y poder abarcarlo mejor. Estas tareas son las siguientes:

1. Creación del dataset
2. Buscando la mejor combinación
3. Paralelismo
4. Análisis de resultados

Como vemos, la primera tarea está orientada a la creación del dataset. Esto, como ya hemos mencionado anteriormente, se debe a la importancia de tener un buen dataset, ya que de ello depende en gran parte los resultados que se obtengan.

IMPLEMENTACIÓN

TECNOLOGÍAS EMPLEADAS

Primero, para comenzar con los pasos que hemos indicado, deberemos preparar el entorno.

Para ello, necesitaremos trabajar sobre un sistema operativo Ubuntu; en este caso trabajaremos sobre la versión 16.04 LTS. Dentro del sistema, iremos instalando las tecnologías que vayamos necesitando. En concreto, de momento es necesario instalar Python 2.7 que será el lenguaje que usaremos en el proyecto. Actualmente de este lenguaje existen dos versiones diferentes en uso, pero nosotros usaremos la que hemos indicado. Para instalarlo, seguiremos el manual que tenemos disponible en el siguiente enlace.

http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/

Las librerías de Python que necesitaremos son:

- OpenCV, la utilizaremos en su versión 3.2. En el anterior enlace también nos indica como instalar esta librería que es una de las más importantes y usadas en el proyecto. Como es una de las librerías fundamentales del proyecto, dedicaremos la siguiente sección a explicarla brevemente.
- Sklearn es una librería orientada al aprendizaje automático en Python. Al estar orientada al machine learning tiene funciones para ayudarnos en la reducción de la dimensionalidad o para seleccionar un modelo. De esta librería, nosotros usaremos las funciones que podemos encontrar dentro del paquete metrics que a su vez está dentro del paquete model selection. Aunque sirva para muchas cosas más, nosotros la vamos a utilizar para realizar la división del dataset de nuestro proyecto de forma automática. Dependiendo del tamaño del dataset y del proyecto que se esté desarrollando, necesitaremos dividirlo de una forma u otra. Las divisiones más comunes son: 90% de entrenamiento y 10% para test, 75% entrenamiento y 25% test o 66% entrenamiento 33% test. Además de hacer la división del dataset de esta forma, también nos permite hacerlo con validación

cruzada, que es la que finalmente se usará en el algoritmo desarrollado, y tiene métodos que nos proporcionan estadísticas muy importantes a partir de los resultados de las comparaciones. Para obtener toda la información de esta librería, tenemos la documentación disponible en su página web (http://scikit-learn.org/stable/index.html).

- Pandas. Es una librería para Python muy útil que nos permite leer, escribir y trabajar con archivos CSV. La interfaz que presenta para el manejo de este tipo de archivos es bastante simple y fácil de usar que tener que recorrer el archivo entero hasta encontrar el dato que se necesita.
- Multiprocessing. Esta librería nos ayudará a la hora de realizar la paralelización del programa.

Además, también necesitaremos otros programas y un entorno de desarrollo. Necesitaremos un editor de imágenes, que en nuestro caso será ImageJ para Windows, porque ya lo tenemos instalado, pero valdría cualquier otro editor, como por ejemplo Gimp, ya que solo lo necesitamos para hacer recortes, tal y como veremos más adelante.

También se instalará un IDE de desarrollo para Python, como es Pycharm, que cuenta con dos versiones, una de pago y otra gratuita. En este caso instalaremos la versión gratuita, porque para lo que necesitamos hacer es más que suficiente.

OPENCV

Antes de empezar a desarrollar el proyecto, conviene explicar en profundidad una de las tecnologías más importantes que se emplearán durante el mismo. Se trata de una librería de código abierto dedicada a la visión por computador y al procesamiento de imagen digital, puesto que necesitaremos modificar algunas de nuestras imágenes para poder trabajar luego con ellas. A lo largo del documento, iremos viendo ejemplos en los que utilizaremos OpenCV. Esta librería contiene los algoritmos que se han comentado para detectar los puntos claves, calcular los descriptores de las imágenes y realizar los *matching* entre las imágenes. Además, con la API que proporciona, podremos realizar diferentes modificaciones sobre las imágenes con pocas líneas de código.

Para entenderlo rápidamente, la visión por ordenador o visión artificial es un campo de la inteligencia artificial orientado a que los ordenadores puedan extraer información de

las imágenes o vídeos para dar solución a ciertos problemas. Se intenta que los ordenadores sean capaces de "ver". Esto, que para nosotros es un proceso sencillo, resulta un proceso bastante más complejo al realizarlo en un ordenador.

OpenCV fue desarrollada inicialmente por la empresa Intel para crear un entorno de desarrollo fácil de usar y que, además, sea altamente eficiente. Lo encontramos desarrollado en C++, aunque tiene interfaces para más lenguajes como Java y Python. Además de estar disponible para varios lenguajes, se encuentra disponible para varios sistemas operativos: Windows, Linux, Mac OS y también para sistemas operativos móviles como iOS o Android.

Existen varias versiones de la herramienta. Se comenzó a usar la versión 2.4 en el proyecto, pero hubo que actualizar porque en el clúster de servidores de Beronia la versión instalada es la 3.2 y la interfaz de las versiones no es la misma.

CREACIÓN DEL DATASET

OBTENER LOS DISCOS

Ahora que tenemos configurado el entorno sobre el que trabajaremos, podemos empezar con la creación del dataset.

El primer paso es recoger los discos de las imágenes que tenemos disponibles. Para ello usaremos un programa que nos proporcionó el profesor Jónathan Heras para obtener los discos a partir de las imágenes de los discos de Petri que hemos visto antes. Antes de utilizarlo, se ha estudiado y se ha modificado para conseguir que los círculos detectados en lugar de ser mostrados se guarden en el disco duro. Debido a la variabilidad de las imágenes, al estar hechas con diferentes dispositivos y condiciones de luz y distancias como ya hemos comentado, el resultado no es siempre el que queremos. A veces los discos son demasiado pequeños y no se detectan; otras veces no son detectados debido a las condiciones de luz. En otras ocasiones se guardan imágenes que no corresponden a discos, por lo que se tienen que borrar. Finalmente, para todas estas excepciones tendremos que guardar estos discos a mano.

Es aquí donde tendremos que usar ImageJ, uno de los programas que acabamos de nombrar, para realizar esa tarea. Simplemente tendremos que ir recortando manualmente los discos que falten de las imágenes de los antibiogramas.

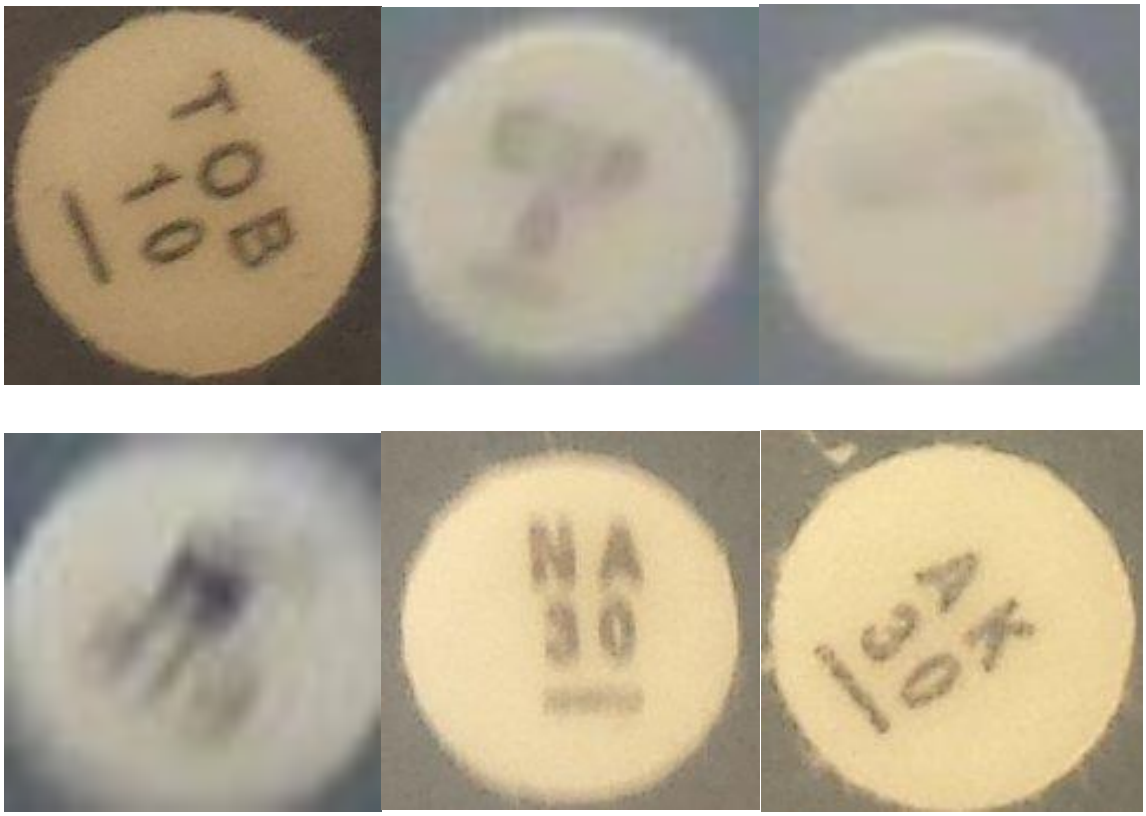

Ilustración 8. Ejemplos de discos de antibióticos

El resultado de este proceso es un conjunto de 311 imágenes que hay que etiquetar y clasificar para seguir con el proceso de creación del dataset. Una vez ya tenemos todos los discos, miraremos el tamaño de las imágenes. Necesitamos que todas las imágenes tengan unas dimensiones similares. Para conseguirlo, tendremos que ampliar los discos que sean más pequeños. Esta tarea, de nuevo, la podemos hacer con cualquier editor de imágenes. Nosotros continuaremos utilizando ImageJ.

Este es el primer paso en la creación del dataset. Este banco de imágenes es muy importante ya que, a partir de un buen banco de imágenes, obtendremos mejores resultados en las pruebas y, en el entrenamiento del modelo, ayudará a que generalice sobre las nuevas imágenes mejor. Al final, puede ayudar a que los resultados que se obtengan sean mejores. Con este primer paso hemos recogido todas las imágenes que

formarán parte del dataset, pero todavía debemos seguir varios pasos para que quede completo.

AGRUPAR LOS DISCOS

Ahora, tenemos todos los discos del mismo tamaño, y ya hemos revisado que todos se han guardado correctamente y que no hay imágenes que no sean de discos. Una vez realizado esto, podremos asignarles las etiquetas a las imágenes. Aquí hay que tener cuidado, ya que en algunas imágenes el tipo de los discos no se ve demasiado bien y podríamos equivocarnos al clasificarlos. Para que esto no pase, se ha decidido no clasificar esas imágenes y descartarlas del dataset, así evitamos estos errores que podrían afectar a los resultados posteriores. Hechos todos los descartes necesarios, creamos las carpetas y clasificamos las imágenes según el tipo que sean. El resultado es una estructura de carpetas como la siguiente (ver ilustración 9):

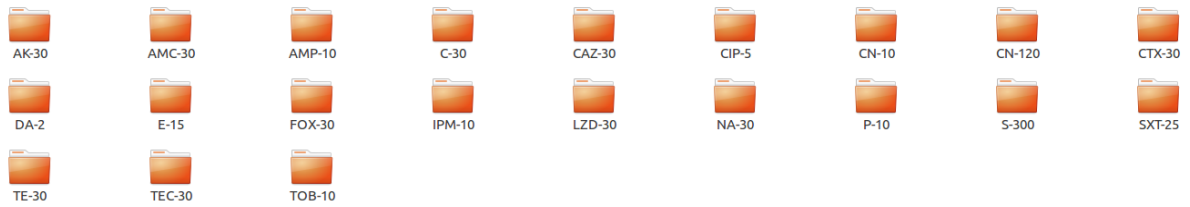

Ilustración 9. Estructura del dataset

Aquí es donde les estamos asignando las etiquetas a las imágenes para poder entrenar al algoritmo y luego comprobar si el tipo predicho es el que se esperaba o no. Las etiquetas deberán venir de un conjunto fijo de categorías. Estas categorías no se pueden añadir ni eliminar una vez que hayamos fijado las categorías que queremos estudiar.

Este paso es también fundamental en el proceso, puesto que sin las etiquetas no podríamos saber si el entrenamiento se ha hecho bien o mal para más tarde poder predecir el tipo de las imágenes que nos vengan.

EXTRAER LETRAS

Este paso podríamos realizar tanto en este momento como antes de agrupar los discos. Con las imágenes clasificadas por tipos, ahora, de nuevo con ayuda de un programa que nos proporciona el tutor, tendremos que recoger de los discos solo las letras que nos indican de qué tipo es cada disco de antibiótico. Entonces, antes de ejecutar el programa para ver qué hace, se estudia y modifica lo necesario para adaptarlo a nuestro caso. Para extraer las letras no se utiliza un proceso de inteligencia artificial como puede ser el de calcular las proyecciones de las imágenes, sino que se emplean métodos de procesamiento de imágenes, cómo es detectar los contornos de las letras. Es conveniente evitar usar procesos complejos y generales de inteligencia artificial porque suelen ser costosos y, en su lugar utilizar técnicas *ad hoc* para el problema concreto. Por ejemplo, en este caso es mucho más fácil y sobre todo rápido obtener las letras de los discos mediante la detección de contornos.

En el siguiente código se observa un fragmento de código del programa. Vamos a explicar qué hace para entenderlo completamente (ver ilustración 11).

Como apreciamos, tenemos que recorrer todas las imágenes dentro de la carpeta que le hayamos pasado como argumento al programa y que estén en formato tif. Luego, recogeremos de la foto que vayamos analizando el nombre y la leeremos para poder así convertirla al modelo de color HSV. También nos interesa recoger las dimensiones de la imagen inicial. Más tarde, con ayuda de una máscara y un método para detectar contornos, nos quedaremos con los contornos para después comprobar si son correctos o no.

Ahora recorreremos todos los posibles caracteres recogidos y comprobaremos si realmente lo son o no. Con los que se han obtenido como verdaderos caracteres, de nuevo, tendremos que recorrerlos y las iremos guardando en una nueva máscara que es la que finalmente nos servirá para quedarnos con los caracteres del disco. Una vez que ya hemos conseguido eso, comprobamos si la ruta de destino pasada por parámetro existe para guardarla y con ello ya guardamos la imagen con los contornos. El resultado de ejecutar el programa que acabamos de ver es el siguiente (ver ilustración 10):

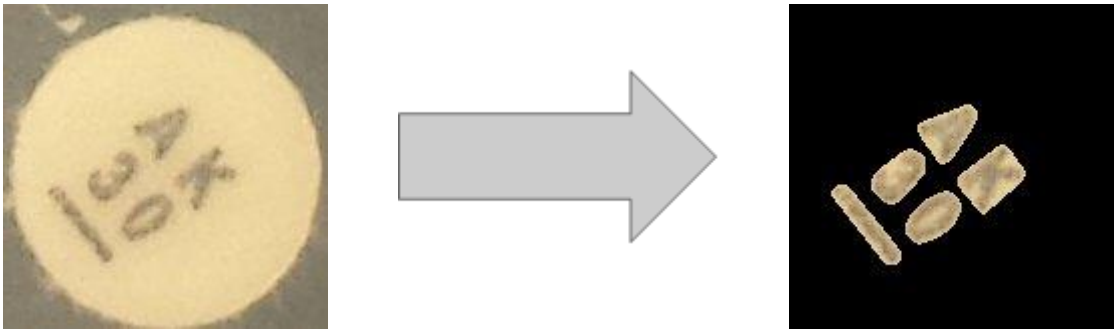

Ilustración 10. Recogida del texto

```
48      for path in glob.glob(args["disks"] + "/*.tif"):
49          name = path[path.rfind("/") + 1:]
50          image = cv2.imread(path)
51          (h, s, v) = cv2.split(cv2.cvtColor(image, cv2.COLOR_BGR2HSV))
52          vCopy = v.copy()
53          (w, h) = image.shape[:2]
54          mask = np.zeros(image.shape[:2], dtype="uint8")
55          cv2.circle(mask, (w / 2, h / 2), (int)(np.minimum(w, h) / 2 * .95), 255, -1)
56          v = cv2.bitwise_and(v, v, mask=mask)
57          v = cv2.adaptiveThreshold(v, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 25, 5)
58          cv2.imshow("im", v)
59          cv2.waitKey(0)
60          (_,cnts, _) = cv2.findContours(v.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
61          charCandidates = np.zeros(v.shape, dtype="uint8")
62
63          goodContours = []
64          for c in cnts:
65              b = cv2.boundingRect(c)
66              print_contour(b, w, h)
67
68              if good_contour(b, w, h):
69                  goodContours.append(c)
70
71          for c in goodContours:
72              hull = cv2.convexHull(c)
73              cv2.drawContours(charCandidates, [hull], -1, 255, -1)
74
75          imageMask = cv2.bitwise_or(image, image, mask=charCandidates)
76          rutaDest = "discosLetras/" + path.split("/")[1]
77
78          if not os.path.isdir(rutaDest):
79              os.mkdir(rutaDest, 0o777)
80
81          cv2.imwrite(rutaDest + "/" + name, imageMask)
82          cv2.imshow("image", imageMask)
83          cv2.waitKey(0)
84
```

Ilustración 11. Código para recoger el texto

AUMENTAR EL DATASET

Hasta el momento, lo que hemos conseguido es sacar los discos de las imágenes que teníamos al principio y, con ellas, organizar los mismos en función de su tipo y quedarnos solo con las letras que contienen. Pero, tras revisar el número de fotografías, hemos observado que son muy pocas las que tenemos - sobre todo de alguno de los tipos que solo tenemos una imagen - y, como ya hemos mencionado, necesitamos un dataset con

un número aceptable de fotografías por clase. Pero, ¿cómo vamos a conseguir más discos si ya hemos recogido todos de los antibiogramas que teníamos? Cierto es que también hemos eliminado algunas imágenes por estar borrosas, pero estas no iban a ayudarnos a obtener la clase esperada.

La respuesta la encontramos en la aplicación de técnicas de aumento (*data augmentation*). Este tipo de técnicas sirven para aumentar el número de imágenes del dataset. Por ejemplo, si giramos la imagen anterior, el tipo del disco sigue siendo AK-30, pero la representación interna (matriz de píxeles) que lo representa habrá cambiado completamente. De esta forma, conseguimos generar más imágenes, que además sirven para cubrir las posibles situaciones en las que nos encontremos nuestros datos. Otros mecanismos que pueden servir para esto son: desplazar la imagen, oscurecerla o cambiar el modelo de color.

El número de fotografías por cada tipo de antibiótico que se quiere obtener es alrededor de 130. Esta es una premisa que debe cumplirse, ya que, de no ser así, al haber una diferencia significativa en el número de ejemplos entre las diferentes clases, es más probable que el resultado obtenido sea el de la clase que tiene mayor número de imágenes, lo que produciría un sesgo.

Por ello, una vez sabido el número de imágenes que se quiere obtener para cada tipo, solo queda, mediante OpenCV, volver a procesar las imágenes para cambiarles el color, rotarlas, hacer traslación, etc, y guardar esas modificaciones en el dataset. A continuación, mostraremos el código necesario para entenderlo correctamente (ver ilustración 12).

En este caso, vamos a recorrer 3 veces las imágenes de la carpeta que hayamos pasado como parámetro. En cada uno de esos recorridos, conseguiremos modificar las imágenes para obtener el número deseado de imágenes por clase.

La primera vez que recorramos las imágenes, las vamos leyendo de una en una y en este caso, les hacemos 3 modificaciones. Primero se oscurecen, para después pasarlas a escala de grises y finalmente ecualizamos el histograma resultante de hacer esas modificaciones. Una vez realizadas las modificaciones, guardamos la imagen resultante con un nombre nuevo para no sobrescribir la imagen original y así aumentar el dataset, que es el objetivo de realizar estas modificaciones.

En el segundo recorrido, el proceso seguido es similar al que acabamos de comentar, pero su función es la de rotar las imágenes. En la captura, el ángulo de rotación es de 153º. Y, por último, la función del tercer recorrido es la de trasladar la imagen. Esto quiere decir que podemos mover la imagen en todas las direcciones, hacia arriba o abajo y hacia la izquierda o derecha.

Una vez que hemos hecho todos estos pasos, habremos acabado la primera parte del diagrama que vimos anteriormente. Después de esto, ya habremos acabado nuestro dataset, puesto que ya tendrá un número de imágenes con el que poder trabajar. Es por ello que ya podemos comenzar el siguiente paso, comparar las imágenes para poder buscar cuál es la mejor combinación.

```
24  i=1;
25  for cosa in listdir(rutaCarpetaDefecto+tipoDisco):
26      print cosa;
27      image = cv2.imread(rutaCarpetaDefecto+tipoDisco+cosa);
28      M = np.ones(image.shape, dtype="uint8") * 50;
29      subtracted = cv2.subtract(image, M);
30      image = cv2.cvtColor(subtracted, cv2.COLOR_BGR2GRAY)
31      eq = cv2.equalizeHist(image)
32      #cv2.imshow("Subtracted", subtracted);
33      cv2.imwrite(rutaCarpetaDefecto + tipoDisco + "subtract"+str(i)  + ".tif", eq);
34      #cv2.waitKey(0);
35      i+=1;
36
```

```
37
38  #Esto es para rotar las fotos
39  i=1;
40  for cosa in listdir(rutaCarpetaDefecto+tipoDisco):
41      if i <55:
42          print cosa
43          image = cv2.imread(rutaCarpetaDefecto+tipoDisco+cosa)
44          rotated = imutils.rotate(image, 153)
45          #cv2.imshow("Rotated by 90 Degrees", rotated)
46          cv2.imwrite(rutaCarpetaDefecto + tipoDisco + "rot"+str(i)  + ".tif", rotated)
47          #cv2.waitKey(0)
48          i+=1
49
```

```
50
51  #Esto es para hacer translaciones de las imagenes
52  i=1;
53  for cosa in listdir(rutaCarpetaDefecto+tipoDisco):
54      print cosa;
55      image = cv2.imread(rutaCarpetaDefecto+tipoDisco+cosa);
56      shifted = imutils.translate(image, 15, 0);
57      #cv2.imshow("Shifted Down", shifted);
58      cv2.imwrite(rutaCarpetaDefecto + tipoDisco + "trans"+str(i)  + ".tif", shifted);
59      #cv2.waitKey(0);
60      i+=1;
61
```

Ilustración 12: Modificación de las imágenes

Una vez hemos ejecutado el anterior algoritmo el resultado obtenido sobre las imágenes es el siguiente (ver ilustración 13):

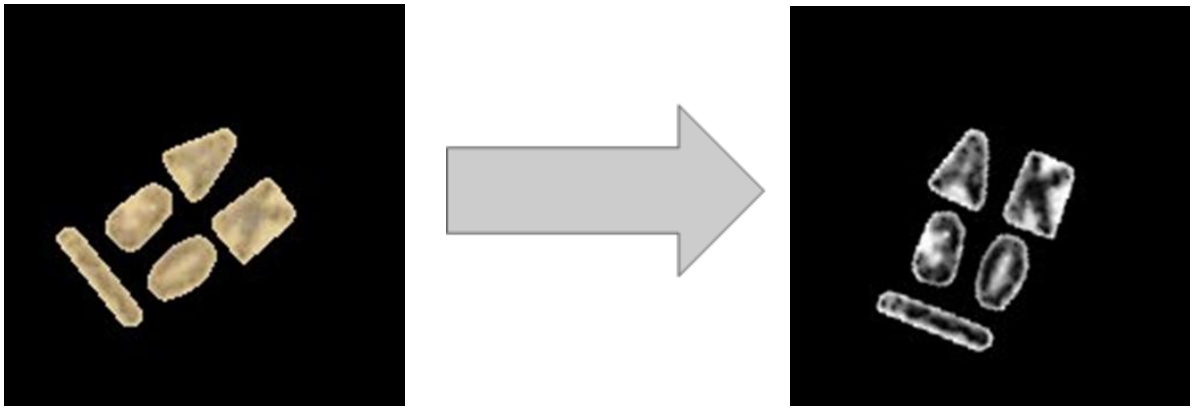

Ilustración 13. Resultado de modificar la imagen

BUSCANDO LA MEJOR COMBINACIÓN

COMPARACIÓN ENTRE DOS IMÁGENES

Dentro de la librería que estamos utilizando, OpenCV, cada tipo de algoritmo desciende de una clase diferente. Los algoritmos para la detección de puntos clave, descienden todos de una clase común de la que necesitamos la función detect. Por su parte, los algoritmos para la extracción de descriptores también tienen una clase padre común, desde esa clase padre tenemos disponible el método compute que usaremos en el proyecto. Finalmente, los algoritmos de *matching* también descienden todos de la misma clase, en este caso, el método que usaremos que es común a todos los algoritmos hijos es knnMatch. Con esta forma de trabajo, conseguimos lo que se llama inyección de dependencias. Esto nos agiliza el proceso de desarrollo puesto que da igual el algoritmo que usemos, porque al descender cada tipo de algoritmo de una clase padre, el método que se utiliza estará disponible en todos los algoritmos del mismo tipo.

Para poder entender correctamente cómo funciona el método de clasificación de una imagen con el dataset, primero tenemos que entender los conceptos más importantes en la comparación entre dos imágenes. De cada una de las imágenes necesitaremos los puntos clave y los descriptores que ya hemos explicado anteriormente qué eran.

Para obtenerlos necesitamos hacer uso de varios métodos y objetos.

1. Tenemos que crear un objeto al que le indicaremos los algoritmos para detectar los puntos clave y generar los descriptores.
2. Ahora con ese objeto al pasarle la imagen que queremos estudiar nos devolverá los puntos clave y sus respectivos descriptores.
3. Con esto, podremos crear un nuevo objeto que es el usaremos para obtener nuestra comparación entre dos imágenes. A este objeto, le pasamos el anterior objeto creado (contiene los algoritmos de detección de puntos clave y genera los descriptores), las imágenes a comparar y el algoritmo de *matching* que se va a utilizar.
4. Gracias a un método de este objeto, podremos realizar la comparación entre las dos imágenes. Ese método, además de toda la información que ya le hemos proporcionado en el constructor, necesita que le pasemos los puntos clave y los descriptores de ambas imágenes. Es entonces cuando con el método de *matching* proporcionado, nos va comparando las imágenes.
5. Finalmente, el resultado de ese método es un porcentaje que nos indica el grado de similaridad que hay entre esas imágenes que lo obtiene a partir del número de puntos clave y descriptores que ha relacionado entre las imágenes.

Este porcentaje se muy importante para poder comparar una imagen con todo el dataset e indicar el tipo del disco que se ha predicho al hacer todas las comparaciones.

COMPARAR IMAGEN CON EL DATASET

Cuando ya tenemos el dataset creado, podemos comenzar a desarrollar la parte dedicada al *matching* de las imágenes y ver las clases que se van obteniendo de las comparaciones. En este paso no vamos a profundizar en la idea del aprendizaje ni en cómo hacer las divisiones del dataset; únicamente haremos una comparación de la imagen contra el dataset para que nos devuelva el tipo que él predice que más se parece. En otras palabras, nosotros tenemos una imagen que queremos comparar con el dataset recién creado, y esa comparación nos debe devolver el tipo del antibiótico que más se asemeja al de la imagen.

El proceso seguido para ello es el siguiente:

1. Comparamos la imagen contra todo el dataset. Esta comparación nos va a devolver un porcentaje indica el grado de similaridad entre las imágenes comparadas. Dichos porcentajes serán guardados para trabajar con ellos.

2. A partir de estos porcentajes, nos quedaremos con los 5 más altos para poder predecir de una forma más segura cuál es el tipo de la imagen que pasamos como parámetro.

3. Ahora, con esos 5 valores más altos, miramos cuál es la clase de cada uno de ellos y el tipo de antibiótico que más veces se repita de esos 5 será el que el método devolverá.

A continuación, vamos a ver el método entero para entender mejor qué es lo que hacemos.

Como el propio código en este caso ya está comentado, no va a hacer falta que hagamos ningún comentario extra por se puede entender fácilmente a que está dedicado cada uno de los métodos utilizados (ver ilustración 14).

```
def similarityImage(imageVector,img, featureDetector, descriptorExtractor, diskMatcher):
    # En esta diccionario guardamos las rutas de las imagenes que han obtenido alto % de coincidencia con la
    # que hemos pasado y el % que tienen.
    matchIm = {}
    maxItems=5 #Este es el numero maximo de elementos que vamos a permitir en el diccionario de los resultados
    queryImage = cv2.imread(img)  # Leemos la imagen pasada como parametro
    dad = DetectAndDescribe(eval(featureDetector),
                            eval(descriptorExtractor))  # Creamos un objeto DetectAndDescribe (creado por Jónathan)
    # al que le pasamos los objetos que detectan los puntos clave de la imagen
    #  y recoge los descriptores a partir de esos puntos clave

    (_, _, v) = cv2.split(cv2.cvtColor(queryImage, cv2.COLOR_BGR2HSV))
    (queryKps, queryDescs) = dad.describe(v);  # En v tenemos la imagen que estamos comparando en cada iteracion con
    # el resto de imagenes de la carpeta
    cv = DiskMatcher(dad, imageVector, diskMatcher) #Comparamos todas las imagenes dentro del vector de imagenes
    #  que le pasamos como parametro
    results = cv.search(queryKps, queryDescs); #Guardamos el vector con todos los resultados de la comparacion
    # de la imagen con las de dentro de la carpeta

    if len(results) != 0:
        for (i, (score, diskPath)) in enumerate(results): #Recorremos el vector de resultados para aniadir solo
            #  las que tengan un % mas alto, el limite de elementos lo ponemos en maxItems
            if (score >= 0.65 ):
                if (len(matchIm) < maxItems): #Solo queremos las maxItems(5) fotos que mas se parezcan a nuestra imagen
                    matchIm[diskPath] = score;
                else:
                    minvalue = min(matchIm.values());
                    if (minvalue < score):
                        minkey = [key for key, value in matchIm.iteritems() if value == minvalue];
                        # Devuelve un vector con los valores minimos, puede ser que sean mas de uno
                        del matchIm[minkey[0]];
                        # borramos del diccionario el primero de los elementos que tenemos guardado en el vector
                        matchIm[diskPath] = score;

```

```
    '''
    Tenemos en matchIm las 5 imágenes que tienen más parecido con la que le hemos pasado por parámetro.
    Con ello ahora tenemos que decir de que tipo se predice que es mirando cual es la clase que mas se repite.
    Para eso crearemos un diccionario que tenga la clase como clave y el número de apariciones como valor,
    así devolveremos la clave que tenga mayor apariciones
    '''
    resultDic={}; #En esta variable guardaremos las rutas que han obtenido alto % de coincidencias
    # y contaremos el número de apariciones para luego devolver la clase que tenga mayor número de apariciones
    for pa, sc in matchIm.iteritems():
        route = pa.split("/");
        if (resultDic.has_key(route[1])):
            resultDic[route[1]]+=1;
        else: resultDic[route[1]]=1;

    if (len(resultDic) != 0):
        maxx = max(resultDic.values());  # cogemos el máximo del diccionario de resultados
        for k in resultDic.items():
            if maxx == k[1]:
                return k[0]
    else: return "Error"
```

Ilustración 14. Comparación de una imagen contra el dataset

Con ayuda de este método, conseguiremos realizar los siguientes pasos que hemos indicado. Para conseguir generalizar este algoritmo a todo el dataset, lo que haremos es llamar a este método para cada una de las imágenes que tenemos dentro del dataset. El problema, como veremos en el siguiente paso, es que no se van a usar todas las imágenes para hacer las pruebas, sino tendremos que usar unas para test y el resto nos servirán para poder comparar esas imágenes de test con el dataset resultante y así probar los resultados que se obtienen.

EVALUANDO LOS ALGORITMOS

Como acabamos de comentar, ahora necesitamos realizar el mismo paso, pero esta vez con todo el dataset. Por ello, lo que haremos será hacer una llamada al método que se acaba de crear; pero, en lugar de hacerlo con todas las imágenes que tenemos el dataset, tendremos que dividirlo en dos partes: por un lado, tendremos la parte de test y por otro el nuevo dataset con el que tendremos que realizar las comparaciones de las imágenes de test.

Con ayuda de la librería sklearn que ya hemos comentado, realizaremos la división. No realizaremos la división del dataset una sola vez para comprobar el funcionamiento del algoritmo, sino que lo realizaremos varias veces mediante un proceso de validación cruzada.

Pero ¿qué es la validación cruzada? Se trata de dividir el dataset en k partes y vamos cogiendo una parte para test y el resto de partes que quedan se usan para entrenar o en

nuestro caso para el proceso de *matching*. Este proceso lo tenemos que repetir hasta que hayamos usado todas las partes para test, por lo que se trata de un proceso largo y costoso. En nuestro caso, como lo hemos dividido en 10 partes, el proceso se repetirá 10 veces, que suele ser el valor más común en estas pruebas.

Ahora, con el dataset dividido en dos partes (para cada una de las validaciones que se realizan), iremos guardando en un vector el tipo real de cada imagen que tenemos en la parte de test. Y, en otro vector, en el que necesariamente se tienen que ir añadiendo los datos en el mismo orden, guardamos el tipo que se ha predicho al usar el método del anterior paso. Una vez realizadas todas las comparaciones, tenemos respectivamente en los vectores los tipos reales y los obtenidos de las comparaciones de las imágenes. Estos vectores los usaremos para generar la matriz de confusión y calcular el accuracy (ratio de acierto) obtenido en esta comparación.

- **La matriz de confusión.** Nos indica de una forma bastante visual los resultados generales obtenidos tras la comparación. Para entenderlo mejor, lo vamos a explicar con un pequeño ejemplo. Tenemos la siguiente matriz con las siguientes clases: *AMC-30, AK-30* y *C-30*. Y se aprecia como tenemos una columna y una fila por clase. Cada elemento de la matriz marca el número de instancias para las cuales la clase real es la indicada por la fila, y la clase predicha por la columna. Para que los resultados sean buenos, como en este ejemplo, los valores grandes tienen que concentrarse en los elementos de la diagonal de la matriz y los valores pequeños fuera de ella. De esta forma se ve claramente cuando un algoritmo tiene muchos errores y las clases para las cuales puede haber más problemas para reconocerlas o, por el contrario, aquellas clases que se han clasificado correctamente.

```
          AMC-30 AK-30 C-30
AMC-30 [[ 2    0     0]
 AK-30  [ 0    8     0]
  C-30  [ 0    0    12]]
```

- **Accuracy (ratio de acierto).** Esta medida, también llamada ratio de éxito global, está definida como el número de clasificaciones correctas entre el número total de clasificaciones.

Para generar las medidas estas medidas, necesitaremos usar la librería sklearn de la cual ya hemos hablado. Para poder generar estas medidas, simplemente tendremos que pasarle a las funciones de la *matriz de confusión y accuracy* los vectores de las clases reales y de las clases predichas y con ellos automáticamente calculará estas medidas.

De momento, esos datos no nos sirven más que para comprobar si hay muchos errores o no en las comparaciones. Más tarde, los utilizaremos para comparar diferentes configuraciones y elegir cuáles son los algoritmos que mejor se adaptan a nuestro problema. Para poder hacer esto, los resultados que se obtengan se recogerán en un archivo CSV y para hacer esto necesitamos la librería pandas que ya hemos comentado en el apartado de tecnologías usadas.

Dentro de este apartado, se encontró un inconveniente cuya solución fue simple. El origen del problema no se encontraba en la implementación del algoritmo a la hora de comparar una imagen contra el dataset, sino que se encontraba en las propias imágenes que estamos utilizando. Parece ser que no todas las imágenes son correctas y que, al realizar las comparaciones entre ellas, originan errores. El error no es nada grave ni crítico y su solución es sumamente simple: solo hay que eliminar la imagen del dataset. Lo complejo o laborioso de esto es saber qué imágenes son las que generan ese error. Para encontrarlas, vamos escribiendo por pantalla la imagen con la que estamos trabajando en cada momento y, cuando veamos que se ha generado el error, sabremos sobre qué imagen es. Tendremos que repetir el proceso hasta ver que todas las imágenes son correctas.

Una vez que hemos acabado de eliminar las imágenes que dan problemas, hay que comprobar el número de imágenes que tenemos ahora en cada una de las clases. En el caso de que hayamos eliminado demasiadas imágenes de una clase, tendremos que generar nuevas imágenes hasta tener de nuevo un número uniforme para todas las clases.

PROBANDO COMBINACIONES DE DESCRIPTORES

Llegando al final del apartado, nos toca comprobar los resultados que nos dan las diferentes combinaciones de descriptores para elegir la que mejor se adapte a nuestro caso y continuar el proceso de *matching*.

Aquí comprobaremos que el dataset se ha creado correctamente ya que si obtenemos problemas para todas las combinaciones eso probablemente será resultado de la creación de un mal dataset.

Antes de probar combinaciones, tendremos que averiguar cuáles tenemos disponibles, para ello, buscaremos en la documentación de OpenCV. Pero esto nos va a generar problemas, porque los algoritmos disponibles en OpenCV son diferentes dependiendo de la versión que se utilice. En la última versión, la que vamos a utilizar que es la 3.2, parte de los algoritmos que vamos a necesitar se encuentran en un proyecto aparte llamado *opencv_contrib*. Instalarlo no es demasiado complejo, pero en contrapartida, no se encuentra instalado en los servidores de Beronia. Entonces, hasta que no se encuentre una forma de tener ese proyecto disponible también en los servidores no podremos trabajar con los algoritmos de ese proyecto. Con este problema profundizaremos más en la sección de paralelización.

Comentar que algunos de los algoritmos no son compatibles con otros, pero se probarán todas las combinaciones posibles para saber qué combinaciones son las que no son compatibles. Los algoritmos que tendremos que combinar son los que hemos explicado anteriormente (detector de puntos clave, extractor de descriptores y algoritmos de *matching*) y tenemos en una la Tabla 1 vista anteriormente.

Este proceso, hay que automatizarlo, ya que tenemos un gran número de combinaciones. En total, 280 combinaciones diferentes si incluimos las que se encuentran en *opencv_contrib*. Además de automatizarlo, es conveniente buscar alguna forma de hacerlo lo más eficiente posible. Haciendo pruebas con un solo proceso y con un dataset pequeño para probar que toda funciona, le ha costado más tiempo del que esperábamos en un principio. Por ello, se ha pensado modificar el algoritmo para conseguir que se ejecute en paralelo y aprovechar al máximo los recursos. Y para conseguir que el tiempo de ejecución sea aún menor, nos han dado acceso a Beronia, que como ya hemos comentado, es un clúster de servidores.

PARALELIZACIÓN

Una vez que hemos probado que el programa funciona, pasamos a la optimización del mismo.

Para ello, lo primero que se pensó fue, en lugar de trabajar en un solo proceso, hacer que el programa se ejecute en varios procesos a la vez como método para solucionar lo comentado en anterior apartado. De esta forma, se optimiza bastante el programa y el tiempo de ejecución se ve reducido. Como en la Universidad de La Rioja está disponible Beronia se pensó hacer uso del mismo para disminuir el tiempo de ejecución del proyecto y para ello, deberemos modificar el programa para que se ejecute en paralelo.

Una vez solucionados todos los problemas que nos encontramos en esta sección, podemos explicar la forma en la que se ha implementado finalmente la paralelización y sobre qué parte del programa se ha realizado para optimizar el tiempo de ejecución. Si nos fijamos en la captura del código (ver ilustración 15), apreciamos que para cada uno de los pasos de la validación cruzada generamos un proceso que guardamos en el vector de procesos al que llamamos *pool*. Para nuestro problema en concreto que hemos dividido este proceso en 10 partes para realizar la validación cruzada, se generan 10 procesos que, al trabajar sobre el servidor, se ejecutarán simultáneamente. Por lo que, para entenderlo correctamente, lo que hemos conseguido ejecutar en paralelo es el proceso de validación cruzada.

```
121     pool = []
122     splits =[]
123     #En esta variable guardamos las variables de todos los pasos para que luego map pueda ejecutarlas de forma paralela
124     manager = Manager() #Objeto que nos permite crear una cola con para compartir los datos entre los procesos
125     queue = manager.Queue()
126     #Creamos una cola donde guardaremos los accuracy que vayan dejando los procesos que ya se hayan ejecutado
127     for train_index, test_index in kf.split(images):
128         varsSet = (train_index, test_index, images, target_names,
129                    featureDetector, descriptorExtractor, diskMatcher,queue)
130         splits.append(varsSet)
131         pool.append(Process(target=calculate_split, args=(varsSet,)))
132
133     for p in pool:
134         p.start()
135
136     while pool:
137         for p in pool:
138             if not p.is_alive():
139                 p.join
140                 if p.exception:
141                     raise Exception
142                 pool.remove(p)
143                 del(p)
144
145     queue.put('DONE') #Para decirle al bucle cuando acabar y salir
```

Ilustración 15. Pool de procesos

HILOS VS PROCESOS EN PYTHON

Pero antes, para poder hacer esto, debemos estudiar cómo funciona el paralelismo en Python. Buscando información en foros encontramos que, por defecto, el uso de más de un hilo en el lenguaje está bloqueado por lo que se denomina GIL (Global Interpreter Lock). Según los comentarios leídos, se debe a que, de este modo, se trata de evitar los problemas que da trabajar con más de un hilo, como ocurre con las condiciones de carrera. Es por ello que se busca conseguir el paralelismo, en lugar de con hilos, con procesos que, aunque sean más costosos de crear, valdrán la pena por la cantidad de datos que tenemos que procesar.

Para poder trabajar con procesos, deberemos importar la librería Python llamada *multiprocessing*. En este caso, no está bloqueado por el propio lenguaje, por lo que tenemos cierta libertad para trabajar de forma paralela con más de un proceso.

Lo primero que haremos será crear un pool de procesos, al que se le puede modificar el tamaño de los procesos sobre los que trabaja. Esto es muy cómodo para hacer pruebas y comprobar que realmente estamos trabajando de forma paralela y poder ver cómo mejoran los resultados en cuanto al tiempo se refiere.

Pero, para conseguir que funcionara, tuvimos que hacer un par de cambios en el código, ya que creaba un objeto que pasábamos como dependencia antes de lo que nosotros deseábamos, lo que originaba errores complejos de solucionar. La solución fue pasar la cadena de texto como parámetro a lo largo de todos los métodos, hasta llegar donde queríamos ejecutarlo; en este punto tuvimos que usar la función *eval* de Python para evaluar la expresión justo donde nosotros deseábamos.

La mejor opción hubiera sido mantenerlo como inyección de dependencias, pero, como ya hemos comentado, daba error.

La solución, según los foros, era hacer un par de cambios algo complejos. Estos cambios no iban a resolver el problema, ya que la solución debe servir también para cuando lo tengamos que ejecutar en el servidor y, esta solución, modificaba una serie de parámetros del sistema que en el servidor no podemos modificar. Por lo que, finalmente, no podíamos seguir usando la inyección de dependencias en esta parte. Por ello, se tuvo que optar por

pasar la cadena de texto hasta el lugar donde teníamos que usar el objeto y ahí, como ya hemos comentado, evaluar la expresión para poder trabajar con ella.

PROBLEMAS CON VERSIONES EN BERONIA

Otro aspecto a comentar es el cambio de versión de la librería de OpenCV. Ahora, necesitamos trabajar con la misma versión que la que tienen instalada en el servidor. Esto es porque, mirando la documentación de OpenCV, al cambiar de una versión a otra los métodos cambian y las funciones de cada versión también son bastante diferentes y las cabeceras de los métodos también pueden cambiar. Por estas razones se ha tenido que actualizar a la versión 3.2, que es la instalada en los servidores. El resultado es que, alguno de los descriptores que usábamos antes, ahora por defecto no están. Para poder trabajar con ellos necesitamos instalar la ampliación que hemos comentado ya un par de veces anteriormente, *opencv_contrib*, en el que sí están implementadas.

Para instalar la versión citada seguimos el mismo manual que hemos indicado al principio del documento (http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/).

Pero, para poder instalar la versión 3.2, primero debemos cambiar algunas cosas. A la hora de hacer el checkout de *opencv* como de *opencv_contrib*, en lugar de hacerlo con la versión 3.0.0 tal y como indica el manual, tenemos que indicar que queremos la versión 3.2.0. Para todo lo demás seguiremos las indicaciones del manual. El proceso es el mismo que se siguió para la instalación al principio del documento.

Antes de probar a ejecutar nada en el servidor, tuvimos que comprobar que la configuración del servidor nos iba a dejar ejecutar sin problemas el proyecto que estamos desarrollando, y que además podemos ejecutar todas las combinaciones de descriptores que hemos preparado. Pero al intentar hacer la primera prueba, vemos como eso no es así y que va a haber que preparar el entorno para posibilitarnos ejecutar todos los descriptores que tenemos planeados. Lo que tendremos que configurar, aparte de tener la misma versión en ambos sitios como acabamos de hacer, es el poder acceder a todos los descriptores ya que en *opencv_contrib* tenemos descriptores importantes como son SIFT y SURF. Puede ser que para nuestro caso no sean los mejores algoritmos, pero

necesitamos comprobar que eso sea así, por lo que necesitaremos añadir de alguna forma estas características al servidor. ¿Cuál es la solución que se ha intentado? A la hora de instalar OpenCV en nuestro equipo, uno de los pasos finales es hacer un enlace a la librería cv2.so. Por ello, se plantea llevarlo a esa librería, ya que está compilada en el servidor. Esto no es una prueba que se hace a ciegas, sino que, para que podamos pensar esto, primero hemos llevado la librería a varias rutas en nuestro equipo en local y se ha intentado trabajar con OpenCV, viendo que efectivamente funcionaba. De ahí que ahora intentemos llevarlo al servidor. Copiamos nuestra librería, que previamente hemos instalado en un sistema operativo Ubuntu 16.04, y probamos a ver si en el servidor funciona todo correctamente.

Tras esto, observamos un error que pide otra librería. La buscamos en nuestro sistema y realmente está, por lo que la añadimos también. Pero, lejos de haber terminado con el problema, vemos otro error que pide una nueva librería. Este problema se repite hasta que llegamos al punto en el que problema ya no es la librería en sí, sino la versión de una de ellas. Buscamos por foros y encontramos que las librerías son demasiado cercanas al sistema operativo y no podremos modificarlas fácilmente, por lo que habrá que buscar otra solución.

Ahora la solución pasa por realizar el mismo procedimiento, pero a partir del mismo sistema operativo que está instalado en los servidores de Beronia, Centos 7.3. Para ello, primero deberemos descargarnos la imagen de instalación del sistema operativo e instalarlo. La instalación no tiene ninguna complicación, solo hay que configurar el lenguaje y parámetros similares. Lo diferente con respecto al procedimiento anterior es la instalación de OpenCV. En este caso, como no tenemos el gestor de paquetes apt-get, hay que instalarla mediante yum. El proceso que se sigue es bastante parecido al que ya hemos comentado, por lo que no lo comentaremos de forma demasiado exhaustiva. Sólo puntualizar que hay que recordar instalar la versión 3.2 cuando hagamos el checkout de los proyectos de OpenCV. En el siguiente enlace tenemos la guía paso a paso:

http://www.computervisiononline.com/blog/install-opencv-31-and-python-27-centos-7

Ahora, con OpenCV instalado, también podremos volver a empezar con la tarea de ir subiendo las librerías que se pidan para hacer funcionar las partes que no están incluidas

por defecto. De nuevo, después de subir unas cuantas librerías, llegamos a un punto en el que nos pide una que no se encuentra instalada. En este caso, la buscamos de nuevo y la podemos descargar para subirla al servidor. Todo hace ver que la librería funciona, pero nos pide un par de ellas más. El proceso que seguimos es igual. Cuando las subimos, por fin no pide ninguna librería más. Por ello, llega el momento de probar que realmente las funcionalidades que queremos se pueden usar y al hacer la prueba vemos que podemos crear los objetos sin ningún tipo de problema. Ya está todo preparado para hacer las pruebas con el servidor. Las páginas en la que encontramos las librerías son las siguientes:

http://rpmfind.net/linux/rpm2html/search.php?query=%2Fusr%2Flib64%2Flibmp.so.3.1.14

https://centos.pkgs.org/6/epel-x86_64/nettle-3.2-2.el6.i686.rpm.html

https://centos.pkgs.org/6/epel-x86_64/gnutls30-3.5.8-1.el6.x86_64.rpm.html

Finalmente apreciamos como el problema está resuelto a pesar del trabajo que ha llevado.

Una vez que se ha probado a usar alguno de los descriptores que faltaban y que no han generado errores podemos pensar en seguir realizando las pruebas de nuestro programa para comprobar si finalmente se puede subir a Beronia para realizar la ejecución y estudiar los resultados.

PROBLEMAS CON LA GESTIÓN DE MEMORIA DE LOS PROCESOS

Los primeros inconvenientes con el código se encontraron antes de mandar nada a ejecutar al servidor, al realizarse las pruebas en local para que, en el caso de obtener errores, se pudiera ver más información sobre los mismos. Y es aquí donde nos damos cuenta de que, después de cierto tiempo, el programa empieza a indicarnos en el archivo donde guardamos los resultados que las combinaciones no son compatibles. Esos errores llaman bastante la atención, ya que se comprueban por separado si realmente esas combinaciones son compatibles o no. Sorprendentemente, la mayoría de los resultados que se han obtenido como no compatibles realmente sí que lo son, por lo que tenemos que investigar dónde ha aparecido ese error. Primero, para obtener más información

sobre el error, hacemos que se muestre más información del mismo en el lugar donde controlamos la escritura en el archivo. Al hacer esto, vemos algo que realmente es bastante llamativo, y es que, según nos indican las trazas de las excepciones que acabamos de controlar, el sistema indica que no puede reservar memoria. Por lo que parece que el problema está en que nos quedamos sin memoria.

Ya tenemos algo más de información sobre el error. Buscando información sobre lo que nos puede estar pasando, encontramos que, en este caso, el error puede estar localizado en el uso de la librería para el pool de procesos. Según los foros consultados, aunque Python tiene un recolector de basura y va cerrando automáticamente los procesos, parece que no lo hace como debería y los va dejando abiertos hasta agotar los recursos del sistema. Esto lleva a que, finalmente, aparezca nuestro error. Para solucionarlo, parece ser que no hay más que cerrar el pool de procesos.

Cuando volvemos a ejecutar las pruebas, todo hace ver que va a funcionar, aunque vuelven a surgir los mismos errores, vemos que aparecen más tarde, por lo que se ha mejorado. Ahora, volvemos a investigar cuál puede ser la causa de nuestro error, pero en este caso no hemos encontrado nada que pueda solucionar nuestro problema y que sirva para llevarlo al servidor. Los foros dicen que se podría aumentar de alguna forma la memoria, pero esa solución no es válida en este caso, ya que no se podría llevar a cabo en el servidor.

Haciendo pruebas para encontrar una solución a este error, probamos a disminuir el número de procesos que se usarán en el pool. Lo reducimos hasta 4 que son los que tenemos en nuestro ordenador. Volvemos a hacer las pruebas y, en este caso, contra todo pronóstico, el programa se ha ejecutado sin ningún tipo de problema, por lo que lo ejecutamos de nuevo para ver si el resultado anterior ha surgido por casualidad o es que esa es la fuente de los problemas. De nuevo, se ejecuta correctamente por lo que se intuye que, al introducir un número demasiado grande de procesos, se acaba quedando sin memoria. Todo hace ver que la librería multiproccesing no está demasiado optimizada y tiene un bug que no nos permite usarla correctamente. Tendremos que tener cuidado con la configuración que le pongamos. Esta, evidentemente, no es la solución esperada. En algunos casos, también surge el problema de la falta de memoria. Entonces hay que seguir buscando una solución para optimizar al máximo posible el uso de la memoria.

Como ya hemos comentado, el problema parece ser que la librería multiprocessing tiene un bug respecto a la optimización del uso de la memoria y no se cierran correctamente los procesos. Esto es algo que se debería hacer automáticamente, aunque el error indica que eso no funciona bien. Entonces, intentamos cerrar los procesos que se vayan creando, pero dentro del pool de procesos no es posible.

Una posible solución a esto es crear un vector de procesos donde se añaden los procesos manualmente (en este caso crearemos 10 procesos que son los splits que tenemos configurados). Luego, recorrer el vector para ir iniciándolos y, cuando se hayan ejecutado, cerrarlos para dejar libre la memoria que estaban consumiendo. Con esto también conseguimos un mayor control sobre los procesos que creamos, llegando a cerrar manualmente los procesos, cosa que hasta ahora no podíamos hacer. La principal consecuencia que conseguimos con esto es liberar la memoria que consumen los procesos y que con el pool de procesos parece que no se liberaba bien.

Después de buscar información al respecto, no hubo una solución clara al problema y se pensó en implementar esa solución. Aunque la solución parezca algo simple, tendremos mayor libertad sobre los procesos que creemos y eso nos ayudará a cerrarlos cuando ya hayan realizado la tarea que queremos. Esto nos ayudará a controlar el uso de memoria que acaban consumiendo por lo que puede ser la solución final a nuestro problema.

Yendo directos a la solución, para implementarla, hay que crear un vector con todos los procesos y arrancarlos manualmente para luego tener control total sobre ello y poder cerrarlos para liberar la memoria. La idea es buena y encontramos un foro donde pudimos ver un ejemplo.

http://www.juanmitaboada.com/multiprocessing-python/

A partir de lo que encontramos ahí, adaptamos nuestro código para poder usar la idea que estamos comentando. Después de las modificaciones nuestro código queda de la siguiente forma (ver ilustración 16):

```
121     pool = []
122     splits =[]
123     #En esta variable guardamos las variables de todos los pasos para que luego map pueda ejecutarlas de forma paralela
124     manager = Manager() #Objeto que nos permite crear una cola con para compartir los datos entre los procesos
125     queue = manager.Queue()
126     #Creamos una cola donde guardaremos los accuracy que vayan dejando los procesos que ya se hayan ejecutado
127     for train_index, test_index in kf.split(images):
128         varsSet = (train_index, test_index, images, target_names,
129                    featureDetector, descriptorExtractor, diskMatcher,queue)
130         splits.append(varsSet)
131         pool.append(Process(target=calculate_split, args=(varsSet,)))
132
133     for p in pool:
134         p.start()
135
136     while pool:
137         for p in pool:
138             if not p.is_alive():
139                 p.join
140                 if p.exception:
141                     raise Exception
142                 pool.remove(p)
143                 del(p)
144
145     queue.put('DONE') #Para decirle al bucle cuando acabar y salir
```

Ilustración 16. Pool de procesos

Donde apreciamos que ahora nuestro *pool* es simplemente un vector en el que guardaremos los procesos que queremos ejecutar más tarde. Estos procesos se encargarán de ejecutar cada uno de los "split" con las variables correspondientes en cada momento.

Una vez tenemos todos los procesos en el vector, los vamos iniciando y, después, vamos comprobando en el vector si ha acabado alguno. Cuando uno haya acabado, se elimina del pool y lo borramos para liberar la memoria y poder trabajar con los siguientes procesos.

Con esto, lanzamos las pruebas primero sobre un dataset pequeño y en local para ver si finalmente el problema de la memoria se ha arreglado o no. La primera ejecución la realiza sin problema. Hacemos lo mismo, pero ahora sobre el servidor para asegurarnos de que funcione y todo hace ver que sí, que funciona sin problemas. Finalmente, a falta de hacer una prueba sobre el dataset real parece que por fin la memoria se gestiona mejor que antes y se puede ejecutar nuestro problema. Ahora si queremos optimizar aún más el programa podemos intentar hacer que la ejecución de las combinaciones de descriptores también se haga en paralelo, pero primero vamos a continuar con las otras tareas.

FALLOS EN LA CREACIÓN DE LOS ARCHIVOS CSV

Solucionado el que probablemente es el problema que más tiempo nos ha dedicado, intentamos continuar con las pruebas para poder ejecutar el proyecto y estudiar los resultados. Pero ese no es el único problema con el que nos hemos encontrado. Aunque se ha solucionado el problema con los procesos, ahora nos damos cuenta de que no se escriben los datos en los archivos correctamente. Tenemos un archivo donde escribimos los tiempos con su combinación correspondiente, otro en el que se escriben los resultados obtenidos junto con la combinación asociada y un último en el que guardamos aquellas combinaciones que no son compatibles. Es el archivo de las combinaciones erróneas en el que encontramos el problema, puesto que no se creaban ni añadían las combinaciones fallidas. Esto se debe a que, al cambiar la forma de trabajar con los procesos, hemos cambiado el "*pool*" de procesos por un vector al que le vamos añadiendo los procesos, y parece que el lugar en el que salta la excepción cuando una combinación no es correcta es diferente, por lo que no la podemos recoger como hasta ahora.

Revisando el programa, logramos controlar la excepción y la vamos propagando hasta el lugar en el que la tenemos controlada al principio. Pero con esto no conseguimos dar solución al problema, por lo que pasamos a tratar de controlar la excepción en el lugar en que se produzca. El problema que encontramos al realizar esto es que añadimos varias veces los datos a los archivos. Al estar trabajando en paralelo, si por ejemplo una combinación no es válida, entramos 10 veces, y como para ninguna de las imágenes se puede hacer (porque la combinación de descriptores es la misma para esas imágenes), salta la excepción y se añade al archivo. Y así con cada uno de los procesos que dan error. De nuevo, vamos a tener que acudir a los foros para encontrar la fuente de nuestro problema. En el siguiente enlace encontramos la solución al mismo.

http://stackoverflow.com/questions/19924104/python-multiprocessing-handling-child-errors-in-parent

El error en la introducción de los datos en los archivos se debe a que los procesos en Python, por defecto, no lanzan excepciones. La solución que encontramos en este caso en stackoverflow es bastante simple, utilizar la noción de clase que tanto conocemos y hacer una nueva clase que herede de la clase Process. En esta nueva clase es donde de alguna forma controlamos si se ha generado o no la excepción.

La clase que hereda de Process queda de la siguiente forma (ver ilustración 17):

```
23  class Process(mp.Process):
24      def __init__(self, *args, **kwargs):
25          mp.Process.__init__(self, *args, **kwargs)
26          self._pconn, self._cconn = mp.Pipe()
27          self._exception = None
28
29      def run(self):
30          try:
31              mp.Process.run(self)
32              self._cconn.send(None)
33          except Exception as e:
34              tb = traceback.format_exc()
35              self._cconn.send((e, tb))
36
37      @property
38      def exception(self):
39          if self._pconn.poll():
40              self._exception = self._pconn.recv()
41          return self._exception
```

Ilustración 17. Clase Process

La interfaz de esta nueva clase es exactamente la misma que la de la clase Process de la que hereda y, al ponerle el mismo nombre, no tendremos que modificar nada del resto del programa.

Como acabamos de ver en la captura, nuestra clase ahora es capaz de controlar las excepciones correctamente. Entonces, para que no se vuelva a dar el mismo problema, vamos lanzando la excepción hasta donde la tenemos controlada en un principio. De este modo, solo se escribe una vez en el archivo correspondiente. Este nuevo inconveniente con la librería *multiprocessing* nos hace ver las dificultades que nos ha provocado. Es evidente el rango de mejora que tiene para poder trabajar eficientemente, ya que hemos tenido que solventar manualmente una serie de problemas que deberían estar implementados de serie, puesto que otros lenguajes ya lo hacen. Por ejemplo, deberían cerrarse automáticamente los procesos dentro del pool para poder ejecutar otro proceso y, de este modo, evitar quedarnos sin memoria en el caso en el que haya que ejecutar muchos procesos, como ocurre en nuestro caso.

Solucionado este problema, finalmente se pueden lanzar las pruebas sobre el servidor. Para lanzar la ejecución sobre el servidor deberemos indicarle un tiempo de ejecución, después de ese tiempo el programa terminará su ejecución. Entonces, es posible que

durante la primera prueba que se haga no se estime bien el tiempo y haya que repetir la prueba o que el tiempo que le cueste ejecutar todas las combinaciones sea demasiado alto y haya que dividir las pruebas para ir haciéndolas por partes.

Aparte de los problemas, a la hora de buscar una solución se encontraron varias posibilidades diferentes a la que hemos implementado. Como el de aumentar la optimización del programa paralelizando alguna parte más del mismo. Se pensó en un primer momento ejecutar en paralelo también cada de las combinaciones de descriptores consiguiendo así aprovechar al máximo los recursos de los servidores. Pero hicimos la prueba y vimos como eso no era posible, puesto que un proceso que hayamos lanzado no puede lanzar otros subprocesos.

Otra idea que se pensó en cuanto a la parelización fue usar el objeto ThreadPool que está contenido en multiprocessing. Está librería no se había mirado hasta ahora porque hay poca documentación sobre ella y podría arreglar el problema de paralelización. Además, se podría hacer con hilos en lugar de procesos siendo estos más rápidos de crear y se pueden crear hilos dentro de los propios hilos. Pero las pruebas que se hicieron nos daban errores en combinaciones que por todas las pruebas realizadas hasta este momento sabemos que sí son correctas. Por esta razón se descartó usar este objeto.

ANÁLISIS DE RESULTADOS

Con todos los problemas del proyecto arreglados, podemos empezar a ejecutar las combinaciones que tenemos almacenadas para estudiar los resultados y conocer la mejor combinación. En total, el número de combinaciones posibles es de 280 aproximadamente, por lo que conseguir que el proyecto se pudiera ejecutar en paralelo y que además se hubiera conseguido poder hacerlo en Beronia era más que necesario para obtener en un tiempo razonable los resultados. El tiempo de ejecución total del programa ha sido de alrededor 15 días completos, lo que resalta la importancia de paralelizar el proceso y usar el servidor para la ejecución.

Una vez que se ha mandado a ejecutar todas las combinaciones posibles al servidor y se han guardado los resultados en los 3 archivos CSV correspondientes (results.csv,

times.csv y resultsError.csv), podremos estudiar cuál de todas las combinaciones es la mejor en cuanto a resultados y tiempo.

A partir de las conclusiones que obtengamos de ambos archivos, si la combinación que resulta ser la mejor es la misma combinación para ambos archivos (tanto en el de los tiempos como en el de los resultados) la solución es clara, esa será la mejor combinación posible. Pero normalmente esto no suele ser así, por lo que tendremos que valorar cuál reporta los mejores resultados dentro de un período de tiempo razonable. Puede que no compense tener una precisión del 98%, por ejemplo, si el tiempo de procesamiento es 4 veces superior al de la segunda combinación que nos proporciona una precisión del 95%.

La combinación resultante será la que utilizaremos para predecir el tipo de los discos y que podrá ser integrada por el tutor en la aplicación AntibiogramJ, que comentamos durante el análisis del proyecto. También conseguimos con esto otro objetivo, poder hacer, con las funciones que hemos desarrollado, predicciones sobre nuevos discos de antibióticos.

Para poder realizar estos estudios, necesitamos usar un programa dedicado a estudiar los archivos CSV y realizar diferentes operaciones para determinar cuál es la mejor combinación para este tipo de problemas. Además de eso, nos nombra también todas las combinaciones en la que los resultados no son muy diferentes a los que se han obtenido con la mejor. Este programa se llama *statisticalComparison,* su uso es muy sencillo y de nuevo nos lo ha proporcionado el tutor para poder realizar el estudio estadístico. Simplemente tendremos que cambiar el valor de un parámetro para poder realizar el estudio sobre nuestros archivos. En ese parámetro, deberemos poner la ruta donde esté ubicado nuestro archivo con el nombre incluido. Con esto sería suficiente para poder ejecutar el programa y obtener la información que queremos.

A partir de la precisión que tenemos almacenada en uno de los archivos, se calcula la media y la desviación estándar de los resultados de la validación cruzada. Para determinar si los resultados son estadísticamente significativos, realiza un test de hipótesis nula para determinar si elegir una prueba paramétrica o una no paramétrica para comparar los modelos. Para determinar si se usa una prueba paramétrica, se deben satisfacer las condiciones de: independencia, normalidad y heterocedasticidad, que ahora explicamos.

- ***Independencia***. Dos sucesos son independientes cuando el hecho de que uno ocurra no influye en la probabilidad de que ocurra el resto. La independencia se cumple siempre porque estamos realizando el estudio de una ejecución de una validación cruzada en la que los datos se recogen de forma aleatoria y deben ser independientes.
- ***Normalidad***. Se considera normalidad al comportamiento que sigue una distribución normal o Gaussiana. Para comprobar esta condición se pueden aplicar varios tests como los tests de Kolmogorov-Smirnov o Shapiro-Wilk, como los que se usan en este estudio estadístico.
- ***Heterocedasticidad***. La heterocedasticidad ocurre cuando la varianza entre las distintas accuracies no es constante a lo largo de las observaciones. La heterocedasticidad se puede comprobar utilizando los tests de Levene o Bartlett.

Calcula el test de Shapiro-Wilk para verificar la normalidad y el test de Levene para verificar la heterocedasticidad. En ambos casos se rechaza la hipótesis nula por lo que se realiza un test no paramétrico.

Una vez determinado el tipo de test que se realiza, se calcula un test de Friedman al tratarse de un test no paramétrico y tener que trabajar con más de dos algoritmos (el archivo CSV con los resultados tiene más de dos combinaciones que probar). El resultado es que se rechaza de nuevo la hipótesis nula lo que implica que los modelos tienen diferentes rendimientos. Con eso, consigue calcular el resultado para combinación y concluye con que la mejor combinación de algoritmos es: Fast-BRISK-BruteForce-Hamming (Esta notación, se refiere a: *<Detector de puntos clave>-<Extractor de descriptores>-<Algoritmos de matching>*) es la mejor.

Ahora que ya sabemos que combinación es la mejor, se emplea un procedimiento post-hoc para abordar los múltiples test de hipótesis entre los diferentes modelos. Como estamos ante el caso no paramétrico, se calcula un procedimiento post-hoc Holm que nos proporcionará información sobre la comparación entre la mejor combinación sobre cada una de las demás. Además de eso, nos informará si hay diferencias significativas entre la combinación que ha resultado ganadora y el resto de combinaciones. Finalmente calcula Eta cuadrado para el tamaño del efecto que obtenemos como resultado un valor de 0.975927 que el propio estudio nos indica que es grande. El informe completo esta adjunto como un anexo del trabajo.

Por su parte, los resultados que se obtienen en torno al tiempo son que la mejor combinación es Fast-ORB-BruteForce-Hamming(2).

Como vemos, no es la misma combinación en ambos casos. Así que, para poder elegir una que tenga una buena relación entre ambos resultados, vamos a recoger las combinaciones que en el análisis estadístico no tienen una diferencia significativa con la ganadora. Y solo con esas combinaciones vamos a dividirlas entre el tiempo de ejecución de cada una para saber cuál de todas tiene una mejor relación. La combinación con mejor relación eficacia/tiempo será la elegida. Para poder ver bien la relación, tenemos una tabla que nos muestra la combinación con su precisión tiempo y la relación entre ambas y, además, están ordenadas por la relación obtenida. Como se aprecia aquí, la combinación que tiene mayor precisión es la 6ª en cuanto al tiempo, por lo que sería más conveniente utilizar alguna de las que se encuentran en los primeros lugares. La diferencia en la precisión no va a ser un problema, el estudio estadístico nos ha dicho que no había grandes diferencias entre las combinaciones que encontramos en la siguiente tabla (ver tabla 2).

Ahora, después de hacer esta comparativa, podemos decir que la mejor combinación precisión tiempo es Fast-FREAK-BruteForce-Hamming(2).

Estos resultados tendrán que ser estudiados por el tutor para que se decida cuál de las combinaciones que tenemos es la que conviene utilizar en el programa AntibiogramJ. Actualmente, según encontramos en el artículo citado en el documento, AntibiogramJ: A tool for analysing images from disk diffusion tests, se está utilizando ORB que en la tabla vemos que no es de la mejores. Entonces se intuye que, a raíz de este estudio, la combinación usada se modifique a la que se ha obtenido como mejor para mejorar el funcionamiento de la aplicación.

	Combinación	Accuracy	Tiempo (seg)	Relación precisión/tiempo
1	Fast-FREAK-BruteForce-Hamming(2)	0,9959893829	3393,90160704	0,0002934644
2	Fast-FREAK-BruteForce-Hamming	0,9970829463	3442,55387592	0,0002896347
3	Fast-FREAK-BruteForce	0,9843185136	3687,68608212	0,0002669204
4	Fast-BRISK-BruteForce-Hamming(2)	0,9974479098	3822,2266531	0,0002609599
5	Fast-FREAK-BruteForce-L1	0,9927033842	3807,15550804	0,0002607467
6	Fast-BRISK-BruteForce-Hamming	0,9985428003	4175,88022399	0,0002391215
7	ORB-ORB-BruteForce-Hamming(2)	0,9839562044	4321,51860189	0,0002276876
8	Fast-BRISK-BruteForce-L1	0,9967193099	4526,31608796	0,0002202054
9	ORB-ORB-BruteForce-Hamming	0,9810431321	4548,89558792	0,0002156662
10	Fast-BRISK-BruteForce	0,9941645654	4723,36332607	0,0002104781

Tabla 2. Clasificación de las combinaciones de descriptores

CONCLUSIONES Y TRABAJO FUTURO

Como punto final del trabajo, queda por hablar de las conclusiones obtenidas y del trabajo futuro tras la realización del proyecto.

En cuanto a las conclusiones acerca del proyecto realizado, estas son bastante claras. Los objetivos que se habían planteado al principio del mismo han sido logrados. Se ha desarrollado con éxito el estudio de las combinaciones de algoritmos disponibles y se han obtenido como conclusión las que mejores resultados han dado dependiendo del tiempo, del accuracy y en relación accuracy/tiempo. Con ello, como se ha visto a lo largo del proyecto, también se ha conseguido realizar el clasificador de imágenes que ha sido necesario para poder realizar el estudio de las combinaciones de algoritmos. Además, durante el progreso del mismo, se ha hecho frente a multitud de problemas que en un primer momento no habían sido previstos. Sobre todo, a los que más tiempo hemos tenido que dedicar han sido los problemas generados por el pool de procesos de la librería multiprocessing de la que tanto hemos hablado a lo largo del documento. O, por ejemplo, haber tenido que compilar OpenCV en un sistema operativo Centos para poder obtener una configuración lo más próxima a la de los servidores y poder llevar el proyecto donde tenemos los descriptores que no están dentro del proyecto por defecto de OpenCV. Pero estos y el resto de problemas con los que nos hemos ido encontrado han sido resueltos de una u otra manera. Estos problemas también han sido más frecuentes sobre todo al principio del proyecto. Se debe especialmente a que empezamos a trabajar con un lenguaje como Python con el que hasta el momento no se había trabajado. Entonces como es de esperar, hasta que no te adaptas a este lenguaje y sabes cómo trabajar con él te surgen las dudas básicas y muchos más problemas que una vez se ha avanzado en el proyecto como se ha dado en este caso.

Es por eso que estamos satisfechos con el resultado del proyecto porque se han utilizado tecnologías que hasta ahora en el máster se habían visto pero sin entrar en demasiado detalle. Además, estas tecnologías se han tenido que conectar para obtener los resultados, como puede ser utilizar la computación de alto rendimiento (HPC) con técnicas del aprendizaje automático para que lo cálculos se realicen antes.

Las sensaciones con las conclusiones personales del proyecto van muy en la línea de las del proyecto. En general, estoy más que satisfecho con el resultado obtenido. Se han logrado los objetivos propuestos y, además, los resultados obtenidos se emplearán en el desarrollo de otro proyecto, en el cual deben clasificar los discos de antibiótico. Se han empleado tecnologías y técnicas que han sido trabajadas en el máster, como son la computación de alto rendimiento, el procesamiento de imágenes y el aprendizaje automático. Algunas de estas tecnologías están a día de hoy muy de moda, como es el caso del aprendizaje automático con el tema de los asistentes personales de empresas como Google, Samsung o Apple entre otras muchas.

A lo largo del proyecto, he aprendido cómo se realiza un proceso de aprendizaje automático y a llevarlo a cabo. Este tema en concreto me ha suscitado el interés suficiente como para ir viendo ciertas cosas por mi cuenta y asistir a eventos relativos a este tema. Es realmente interesante el conseguir que una máquina "aprenda" a partir de la configuración de ciertos parámetros y pueda predecir comportamientos futuros con el aprendizaje que ha obtenido del algoritmo que se ha ido utilizando.

Por último, a parte del aprendizaje que podríamos considerar más académico debo destacar el aprendizaje personal. En cuanto a esto, considero que, tras la realización del proyecto, he logrado incrementar mi nivel de autonomía para solucionar los errores que puedan aparecer dentro de proyecto. Ya que, en muchos de los casos, se ha conseguido solucionar el problema encontrado de forma autónoma y en los casos que no ha sido así, antes de preguntar por las alternativas para una posible solución, se han investigado para poder proponerlas y estudiar cuál puede ser la idea que mejor solucionara ese problema encontrado. También la importancia de la comunicación en un proyecto, porque para esas ocasiones en las que teníamos dudas, es conveniente compartirlas para buscar una solución lo antes posible porque si no lo más probable es que retrases el proyecto y no consigas ningún resultado.

Como trabajo futuro queda encontrar alguna mejora en cuanto a la paralelización del programa. Aunque se haya adoptado para el proyecto el paralelizar la validación cruzada (split) de cada una de las combinaciones, puede ser que haya alguna otra parte donde la paralelización sea más conveniente y nos ofrezca mejores resultados. Por ejemplo, hacer que se ejecuten en paralelo las combinaciones de descriptores y no los pasos (split) de la

validación cruzada dentro de cada combinación. El problema de esta solución es que puede ser que hubiera que controlar las condiciones de carrera que se pudieran dar para intentar escribir todos los procesos sobre el mismo archivo CSV. Pero, si no se encuentran problemas con ello o se pueden controlar fácilmente, podría tratarse de una solución preferible a la utilizada.

Haciendo referencia al trabajo futuro, una parte importante sería la optimización del programa. Se puede buscar una forma de conseguir que sea más paralelo o que parte de los cálculos no se tengan que ejecutar tantas veces. Por ejemplo, se ha pensado en hacer que el cálculo de los descriptores y puntos clave de la imagen. En lugar de tener que hacerse cada vez que comparemos una imagen con otra, se realice una vez por cada par de *detector de puntos clave* y *extractor de descriptores*. Así, conseguimos que cada vez que queramos comparar los descriptores de esas dos imágenes no haya que calcularlos, sino que puedan ser directamente recogidos del archivo en el que estén guardados.

BIBLIOGRAFÍA

PROBLEMA BIOLÓGICO

- Alonso, C.A., Domínguez, C., Heras, J., Mata, E.,Pascual, V., Torres C. y Zarazaga, M. (2017). *Antibiogramj: A tool for analysing images from disk diffusion tests.* Universidad de La Rioja. Disponible en: http://www.unirioja.es/cu/joheras/papers/antibiogramj.pdf

INTELIGENCIA ARTIFICIAL

- Heras, J. (2017). *Aprendizaje automático*. Inteligencia Artificial. Universidad de La Rioja. Disponible en: https://docs.google.com/document/d/1zJxnOWmd-8bDJ9_cCDUUKjhf-qfY7Ug0R-eI2_m-8TQ/edit#

- Rouse, M. (Junio 2017). *Aprendizaje automático (machine learning)*. Disponible en: http://searchdatacenter.techtarget.com/es/definicion/Aprendizaje-automatico-machine-learning

VISIÓN POR COMPUTADOR

- Heras, J. (2017). *Visión por computador.* Inteligencia Artificial. Universidad de La Rioja. Disponible en: https://docs.google.com/document/d/1IGRGZ8wWyOR2DWYZnxdIk0nUUklsKKKb6urBwOIWKv4/edit#

- OpenCV (Junio 2017). *OpenCv library.* Disponible en: http://opencv.org/

- OpenCV (Junio 2017), *Installation in Linux.* Disponible en: http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation

PARALELIZACIÓN Y COMPUTACIÓN DE ALTAS PRESTACIONES

- Juanmi (Junio 2017). *Multiprocessing: una piscina de procesos en tu Python*. Disponible en: http://www.juanmitaboada.com/multiprocessing-python/
- Beyer, L (Junio 2017). *Python's undocumented ThreadPool*. Disponible en: http://lucasb.eyer.be/snips/python-thread-pool.html
- Stackoverflow (Junio 2017). *Python Multiprocessing: Handling Child Errors in Parent.* Disponible en: http://stackoverflow.com/questions/19924104/python-multiprocessing-handling-child-errors-in-parent
- Universidad de La Rioja (Junio 2017). *Beronia_UR.* Disponible en: http://stackoverflow.com/questions/19924104/python-multiprocessing-handling-child-errors-in-parent
- Mundo geek (Junio 2017). *Threads en Python*. Disponible en: http://mundogeek.net/archivos/2008/04/18/threads-en-python/
- Sarafraz, A (Junio 2017). *Install OpenCV 3.1 and Python 2.7 on CentOS 7*. Computer Vision Online. Disponible en: http://www.computervisiononline.com/blog/install-opencv-31-and-python-27-centos-7
- Rosebrock, A (Junio 2017). *Install OpenCV 3.0 and Python 2.7+ on Ubuntu. Pyimagesearch*. Disponible en: http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/

APÉNDICE

INSTALACIÓN OPENCV

Por esa razón, en el manual nos indican cómo instalar directamente ambas. El manual lo encontramos en el siguiente link y vamos a ir paso por paso en la instalación. El problema que encontramos es que el manual para la versión 2.4 ya no se encuentra disponible, por lo que tendremos que utilizar el manual de instalación de la versión 3.0, realizando un pequeño cambio. Sin embargo, veremos más adelante que, aunque esta fue la versión instalada en un primer momento, finalmente se tuvo que actualizar.

http://www.pyimagesearch.com/2015/06/22/install-opencv-3-0-and-python-2-7-on-ubuntu/

El primer paso es actualizar las librerías y paquetes que tengamos instalados hasta el momento. Para ello ejecutamos los comandos:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

Una vez están los repositorios actualizados, instalamos las herramientas de desarrollo:

```
$ sudo apt-get install build-essential cmake git pkg-config
```

Ahora bajaremos los paquetes para trabajar con distintos formatos de imagen:

```
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev libpng12-dev
```

Y, para poder mostrar las imágenes, necesitamos la librería GTK.

```
$ sudo apt-get install libgtk2.0-dev
```

Instalamos librerías para procesar vídeo y acceder a los *frames* individualmente.

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
libv4l-dev
```

Después, tenemos que instalar unas librerías para optimizar OpenCV.

```
$ sudo apt-get install libatlas-base-dev gfortran
```

A continuación, necesitaremos un gestor de paquetes para Python, para ello instalamos *pip*. Con este gestor tendremos la posibilidad de instalar entornos virtuales para poder trabajar con python.

```
$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py

$ sudo pip install virtualenv virtualenvwrapper

$ sudo rm -rf ~/.cache/pip
```

Una vez que hemos instalado virtualenv, debemos actualizar el fichero *~/.bashrc* añadiéndole lo siguiente:

```
export WORKON_HOME=$HOME/.virtualenvs

source /usr/local/bin/virtualenvwrapper.sh
```

Con esto, podremos usar los entornos virtuales desde cualquier sitio. Pero, para que esos cambios tengan efecto, hay que ejecutar el siguiente mandato:

```
$ source ~/.bashrc
```

Una vez ejecutado, podremos crear nuestro primer entorno virtual desde el que empezaremos a trabajar.

```
$ mkvirtualenv cv
```

Justo después, instalaremos herramientas de desarrollo de Python 2.7 y también instalamos en el entorno virtual la librería numpy.

```
$ sudo apt-get install python2.7-dev

$ pip install numpy
```

Con el entorno preparado, descargamos de GitHub OpenCV en la ubicación que deseemos y hacemos un checkout de la versión 2.4.2.

```
$ cd ~

$ git clone https://github.com/Itseez/opencv.git

$ cd opencv

$ git checkout 2.4.2
```

Justo después tendremos que configurar la compilación. Para ello, necesitamos ejecutar el siguiente comando:

```
$ cd ~/opencv

$ mkdir build

$ cd build

$ cmake -D CMAKE_BUILD_TYPE=RELEASE \

      -D CMAKE_INSTALL_PREFIX=/usr/local \

      -D INSTALL_C_EXAMPLES=ON \

      -D INSTALL_PYTHON_EXAMPLES=ON \

      -D BUILD_EXAMPLES=ON ..
```

Y, con esto, ya podremos compilar OpenCV.

```
$ make -j4
```

Ese paso es el que más tiempo lleva. Si vemos que se ha compilado sin ningún error, podemos continuar instalándolo.

```
$ sudo make install

$ sudo ldconfig
```

Finalmente, para poder usar OpenCV en cualquier parte de nuestro entorno virtual, necesitaremos crear un enlace a la librería *cv2.so* en la carpeta *site-packages* de nuestro entorno virtual *cv*.

```
$ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/

$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

Una vez seguidos todos estos pasos, solo nos quedaría comprobar que la versión instalada es la que nosotros queremos.

Pycharm: Para instalar este IDE no necesitaremos un proceso tan complejo como el que acabamos de ver. Simplemente necesitaremos entrar en su página web (https://www.jetbrains.com/pycharm/download/#section=linux) y darle a descargar; como ya hemos comentado, instalaremos la versión gratuita. Una vez descargado, solo

tendremos que descomprimir el archivo, entrar en la carpeta que se ha descomprimido y desde, una terminal, ejecutar el archivo pycharm.sh que esta dentro de la carpeta bin. Con esto, tendremos ya corriendo nuestro IDE de desarrollo de Python. Ahora solo queda configurarlo para usarlo con el entorno virtual que acabamos de crear.

ImageJ: En este caso, la instalación la haremos en un sistema operativo Windows. Descargamos el programa de la siguiente página web https://imagej.nih.gov/ij/download.html y descomprimimos el paquete. Una vez descomprimido solo tenemos que ejecutar la aplicación, sin necesidad de instalarlo.

APRENDIZAJE SUPERVISADO

Este tipo de aprendizaje, se basa en, a partir de un conjunto de entrenamiento con etiquetas, construir un modelo capaz de predecir la etiqueta de las nuevas instancias. El proceso de aprendizaje se divide en 5 pasos:

- **Estructurar el dataset inicial:** Si trabajamos con este tipo de aprendizaje necesitamos las instancias y la categoría de cada una de las instancias del dataset. Además, es conveniente tener un número uniforme de instancias por categoría. Si no lo cumplimos, es posible que el clasificador tienda a predecir que las instancias son del tipo del que tenemos más instancias dentro de nuestro dataset. Esto produce lo que se llama sesgo, haciendo que el clasificador no funcione como nosotros queremos que funcione.
- **Partir el dataset en dos partes:** Consiste en dividirlo en una parte llamada conjunto de entrenamiento que se utiliza para entrenar el algoritmo y una parte de test para poder evaluar el rendimiento de los algoritmos. Estas dos partes tienen que ser independientes y no se pueden superponer. Hay muchas posibilidades de hacer la división del dataset, pero las más comunes son las que vemos en la imagen (ver ilustración 18).

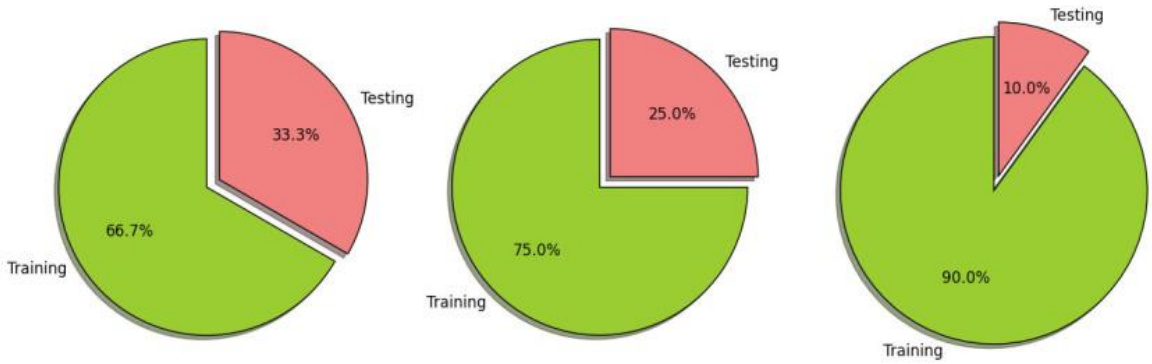

Ilustración 18. Divisiones comunes para un dataset

- **Extraer descriptores:** A partir de cada instancia tenemos que obtener un conjunto de propiedades que almacenamos en un vector que describe esa instancia. Ya sea para el conjunto de entrenamiento o para el de test las propiedades que obtengamos de cada instancia deberán ser las mismas.
- **Entrenar o construir el modelo:** Aquí es donde se utilizan diferentes algoritmos para crear un modelo que sea capaz de predecir las instancias futuras que se le pasen. Los algoritmos que se usan para realizar este paso son: KNN, árboles de decisión, descenso de gradiente, regresión logística, las redes neuronales o SVM entre otros.
- **Evaluar el modelo:** Ahora necesitamos saber cuál de los modelos es mejor para el problema que se estudie. Esto se complica porque no existe un modelo que sea siempre el mejor, sino que se suelen probar varios modelos y además con distintos parámetros y ver cuál de todas las opciones es la que mejor funciona. A este proceso se le llama evaluación. Se usan distintas medidas y procesos para saber cuál es la mejor opción, algunas medidas como las matrices de confusión o las curvas ROC y los procesos que se usan para hacer las evaluaciones como pueden ser la validación cruzada, de la que hablaremos más tarde, o la estratificación entre otras.

A lo largo del proyecto cogeremos las ideas principales de este tipo de aprendizaje. Además, iremos comentando los pasos y explicaremos los conceptos necesarios para entender el funcionamiento del aprendizaje supervisado (Heras, 2017: 13-46).